35th Symposium on Theoretical Aspects of Computer Science

STACS 2018, February 28–March 3, 2018, Caen, France

Edited by

Rolf Niedermeier

Brigitte Vallée

Editors

Rolf Niedermeier
Fakultät IV, AKT
Technische Universität Berlin
rolf.niedermeier@tu-berlin.de

Brigitte Vallée
GREYC, UMR CNRS 6072
Université de Caen Normandie
Brigitte.Vallee@unicaen.fr

ACM Classification 2012
Mathematics of computing, Theory of computation

ISBN 978-3-95977-062-0

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-062-0.

Publication date
February, 2018

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.STACS.2018.0

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Contents

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

Foreword

The Symposium on Theoretical Aspects of Computer Science (STACS) conference series is an international forum for original research on theoretical aspects of computer science. Typical areas are:

- algorithms and data structures, including: design of parallel, distributed, approximation, parameterized and randomized algorithms; analysis of algorithms and combinatorics of data structures; computational geometry, cryptography, algorithmic learning theory, algorithmic game theory;
- automata and formal languages, including: algebraic and categorical methods, coding theory;
- complexity and computability, including: computational and structural complexity theory, parameterized complexity, randomness in computation;
- logic in computer science, including: finite model theory, database theory, semantics, specification verification, rewriting and deduction;
- current challenges, for example: natural computing, quantum computing, mobile and net computing, computational social choice.

STACS is held alternately in France and in Germany. This year's conference (taking place February 28–March 3 in Caen) is the 35th in the series. Previous meetings took place in Paris (1984), Saarbrücken (1985), Orsay (1986), Passau (1987), Bordeaux (1988), Paderborn (1989), Rouen (1990), Hamburg (1991), Cachan (1992), Würzburg (1993), Caen (1994), München (1995), Grenoble (1996), Lübeck (1997), Paris (1998), Trier (1999), Lille (2000), Dresden (2001), Antibes (2002), Berlin (2003), Montpellier (2004), Stuttgart (2005), Marseille (2006), Aachen (2007), Bordeaux (2008), Freiburg (2009), Nancy (2010), Dortmund (2011), Paris (2012), Kiel (2013), Lyon (2014), München (2015), Orléans (2016), Hannover (2017).

The interest in STACS has remained at a high level over the past years. The STACS 2018 call for papers led to 186 submissions with authors from 36 countries. Each paper was assigned to three program committee members who, at their discretion, asked external reviewers for reports. The committee selected 54 papers during a three-week electronic meeting held in November/December. For the third time within the STACS conference series, there was also a rebuttal period during which authors could submit remarks to the PC concerning the reviews of their papers. As co-chairs of the program committee, we would like to sincerely thank all its members and the many external referees for their valuable work. In particular, there were intense and interesting discussions inside the PC committee. The overall very high quality of the submissions made the selection a difficult task.

This year, the conference includes a tutorial. We would like to express our thanks to the speaker Bruno Salvy for this tutorial, as well as to the three invited speakers, Meena Mahajan, Damien Pous, Gerhard Woeginger. Special thanks go to the local organizing committee for continuous help throughout the conference organization. In particular, we wish to thank Nicolas Bedon, Julien Clément, and Julien Courtiel for their work in the edition of the proceedings, and Ali Akhavi in the general organization; we also wish to thank Virginie Desnos-Carreau and Agnès Zannier for their administrative support.

Moreover, we thank Michael Wagner from the Dagstuhl/LIPIcs team for assisting us in the publication process and the final production of the proceedings. These proceedings contain extended abstracts of the accepted contributions and abstracts of the invited talks

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

and the tutorial. The authors retain their rights and make their work available under a Creative Commons license. The proceedings are published electronically by Schloss Dagstuhl – Leibniz-Center for Informatics within their LIPIcs series.

STACS 2018 has received funds and help from the following various institutions, for which we are very grateful:

- CNRS (Comité National de la Recherche Scientifique)
- IUF (Institut Universitaire de France)
- Mathematics and Computer Science Laboratories in Normandy
  - GREYC (Groupe de Recherche en Informatique, Image, Automatique et Instrumentation de Caen): Université de Caen Normandie, ENSICAEN, CNRS;
  - LITIS (Laboratoire d'Informatique, du Traitement de l'Information et des Systèmes): Université de Rouen, Université du Havre, Institut National des Sciences Appliquées (INSA);
  - LMNO (Laboratoire de Mathématiques Nicolas Oresme): Université de Caen Normandie, CNRS;
  - NormaSTIC (Fédération Normande de Recherche en Sciences et Technologies de l'Information et de la Communication)
- Université de Caen Normandie;
- Agglomération Caen-la-Mer.

Caen and Berlin, March 2018 — Brigitte Vallée and Rolf Niedermeier

Conference organization

Program Committee

Shweta Agrawal	Indian Institute of Technology, Madras
Markus Bläser	Saarland University, Saarbrücken
Anne Bouillard	Ecole Normale supérieure, Paris.
Véronique Bruyère	Université de Mons
Arnaud Carayol	Université Paris-Est Marne-la-Vallée
Jérémie Chalopin	CNRS and Université Aix-Marseille
Vida Dujmović	University of Ottawa
Samuel Fiorini	Université libre de Bruxelles
Eric Fusy	CNRS and Ecole Polytechnique, Palaiseau
Paweł Gawrychowski	University of Haifa
Jarkko Kari	University of Turku
Michael Kaufmann	Eberhard Karls Universität Tübingen
Juha Kontinen	University of Helsinki
Daniel Král	University of Warwick
Francois Le Gall	Kyoto University
Christof Löding	RWTH Aachen
Andrew McGregor	University of Massachusetts, Amherst
Stefan Mengel	CNRS and Université d'Artois, Lens
Rolf Niedermeier	Technische Universität Berlin (co-chair)
Dirk Nowotka	Christian-Albrechts-Universität, Kiel
Daniel Panario	Carleton University
Michał Pilipczuk	University of Warsaw
Daniel Reidenbach	Loughborough University
Thomas Sauerwald	University of Cambridge, UK
Sabine Storandt	Julius-Maximilians-Universität, Würzburg
Howard Straubing	Boston College
Szymon Toruńczyk	University of Warsaw
Brigitte Vallée	CNRS and Université de Caen Normandie (co-chair)
Virginia Vassilevska Williams	Massachusetts Institute of Technology, Cambridge, US

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

Local Organizing Committee

Ali Akhavi (Caen), co-chair
Mustapha Arfi (Le Havre)
Nicolas Bedon (Rouen)
Julien Clément (Caen), co-chair
Julien Courtiel (Caen)
Virginie Desnos-Carreau (Caen)
Theo Grente (Caen)
Etienne Grandjean (Caen)
Giovanna Guaiana (Rouen)
Loïck Lhote (Caen)
Florent Madelaine (Clermont Ferrand)
Ayoub Otmani (Rouen)
Carla Selmi (Rouen)
Brigitte Vallée (Caen)
Véronique Terrier (Caen)
Agnès Zannier (Caen)

External Reviewers

Ali Akhavi
Eric Allender
Jean-Paul Allouche
Josh Alman
Klaus Ambos-Spies
Alexandr Andoni
Patrizio Angelini
Srinivasan Arunachalam
Kazuyuki Asada
James Aspnes
Sepehr Assadi
Haris Aziz
Arturs Backurs
Eric Badouel
Cyril Banderier
David Barrington
Nicolas Basset
Paul Beame
Martin Beaudry
Michael Bekos
Djamal Belazzougui
Rémy Belmonte
Shalev Ben-David
Matthias Bentert
Debajyoti Bera
Luca Bernardinello
Dietmar Berwanger
René van Bevern
Olaf Beyersdorff
Anup Bhattacharya
Jean-Camille Birget
Achim Blumensath
Greg Bodwin
Andrej Bogdanov
Adrien Boiret
Marthe Bonamy
Ievgen Bondarenko
Nicolas Bonichon
Édouard Bonnet
Prosenjit Bose
Nicolas Bousquet
Steven Brams
Robert Bredereck
Andrej Brodnik
Andrei Bulatov
Laurent Bulteau
Marc-Olivier Buob
Clément Canonne
Yixin Cao
Florent Capelli
Marco Carmosino
Julien Cassaigne
Steven Chaplick
Arkadev Chattopadhyay
Cédric Chauve
Jiehua Chen
Rajesh Chitnis
Christian Choffrut
Julien Clément
Thomas Colcombet
Gwendal Collet
Basile Couëtoux
Wojciech Czerwiński
Loris D'Antoni
Shantanu Das
Samir Datta
Anuj Dawar
Joel Day
Akshay Degwekar
Volker Diekert
Thomas C. van Dijk
Britta Dorn
Manfred Droste
Amalia Duch
Bartlomiej Dudek
Maciej Dulęba
François Durand
Zdenek Dvorák
Thorsten Ehlers
Andreas Emil Feldmann
Leah Epstein
Alessandro Facchini
John Fearnley
Nathanaël Fijalkow
Paola Flocchini
Till Fluschnik
Henry Förster
Kyle Fox
Dominik D. Freydenberger
Travis Gagie
Anahí Gajardo
Nicola Galesi
Robert Ganian
Shashwat Garg
Gilles Geeraerts
Silvio Ghilardi
Mohammad Ghodsi
Archontia Giannopoulou
Dion Gijswijt
Emmanuel Godard
Thibault Godin
Tomasz Gogacz
Mordecai J. Golin
Petr Golovach
Alexander Golovnev
Daniel Gonçalves
Laure Gonnord
Mika Göös
Erich Grädel
Frederic Green
Daniel Grier
Bernhard Haeupler
Tero Harju
Quentin Hautem
Chinmay Hegde
Lauri Hella
Shuichi Hirahara
Mika Hirvensalo
Åsa Hirvonen
Peter Høyer
Xuangui Huang
Tomohiro I
Yoichi Iwata
Jérémie Jakubowicz
Erik Jan van Leeuwen
Artur Jeż
Felix Joos
Stephen Jordan
Stasys Jukna
Juhani Karhumaki
Juha Kärkkäinen
C. S. Karthik
Dominik Kempa
Eun Jung Kim
Hartmut Klauck
Will Klieber
Bartek Klin
Vladimir Kolmogorov
Balagopal Komarath
Christian Komusiewicz
Eryk Kopczynski

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

Janne H. Korhonen
Sajin Koroth
Dmitry Kosolobov
Mirosław Kowaluk
Marcin Kozik
Artur Kraska
Jan Kratochvíl
Dieter Kratsch
Manfred Kufleitner
Amit Kumar
Marvin Künnemann
Dietrich Kuske
Antti Kuusisto
O-Joung Kwon
Arnaud Labourel
Victor Lagerqvist
Michael Lampis
Patrick Landwehr
Kasper Green Larsen
Sławomir Lasota
Massimo Lauria
Yvan Le Borgne
Thierry Lecroq
Troy Lee
Erik Jan van Leeuwen
Jonas Lefèvre
Karoliina Lehtinen
Aurélien Lemay
Christoph Lenzen
Loïck Lhote
Bernard Lidicky
Nutan Limaye
Didier Lime
Andrea Lincoln
Bruno Loff
Daniel Lokshtanov
Alex Lombardi
Sylvain Lombardy
Steve Lu
Martin Lück
Gábor Lugosi
Marco Macchia
Meena Mahajan
Luidnel Maignan
Andreas Maletti
Florin Manea
Sebastian Maneth
Bodo Manthey
Nicolas Markey
Dániel Marx
Monaldo Mastrolilli
Fabien Mathieu
Jannik Matuschke
Pierre McKenzie
Arne Meier
Hammurabi Mendes
Robert Mercas
Wolfgang Merkle
George Mertzios
Arnaud de Mesmay
Frédéric Meunier
Theresa Migler-Von Dollen
Matúš Mihalák
Malte Milatz
Tillmann Miltzow
Matthias Mnich
Fabio Mogavero
Hendrik Molter
Tobias Mömke
Debajyoti Mondal
Benjamin Monmege
Thierry Monteil
Pat Morin
Antoine Mottet
Priyanka Mukhopadhyay
Aniello Murano
Christopher Musco
Lars Nagel
Masaki Nakanishi
Harikrishna Narasimhan
N. S. Narayanaswamy
Meghana Nasre
Guyslain Naves
Jelani Nelson
Cyril Nicaud
Patrick K. Nicholson
André Nichterlein
André Nies
Damian Niwiński
Ashkan Norouzi-Fard
Krzysztof Nowicki
Pascal Ochem
Joanna Ochremiak
Frédéric Olive
Sebastian Ordyniak
Patric Östergård
Dominik Pająk
Fahad Panolan
Charles Paperman
Ludovic Patey
Daniël Paulusma
Arno Pauly
Guillermo Pérez
Michaël Perrot
Daniela Petrisan
Marcin Pilipczuk
Solon Pissis
Amaury Pouly
Danny Bøgsted Poulsen
Damien Pous
Manoj Prabhakaran
Élise Prieur-Gaston
Daniel Quernheim
Roman Rabinovich
Chrysanthi Raftopoulou
Somindu C. Ramanna
M. S. Ramanujan
Narad Rampersad
Mickael Randour
Jean-Florent Raymond
Ilya Razenshteyn
Oded Regev
Felix Reidl
Klaus Reinhardt
Ling Ren
Renato Renner
Pierre-Alain Reynier
Michel Rigo
Emanuele Rodaro
Jérémie Roland
Panagiotis Rondogiannis
Marc Roth
Jörg Rothe
Christelle Rovetta
Mikhail Rudoy
Aleksi Saarela
Ville Salo
Laura Sanità
Jayalal Sarma
Ignasi Sau
Saket Saurabh
Sven Schewe
Melanie Schmidt
Sylvain Schmitz
Thomas Schneck
Philippe Schnoebelen
Henning Schnoor
Gregory Schwartzman
Helmut Seidl
Olivier Serre

Sebastian Siebertz
Michał Skrzypczak
Friedrich Slivovsky
Christian Sohler
Manuel Sorge
Joachim Spoerhase
Tatiana Starikovskaya
Brett Stevens
Yann Strozecki
Jukka Suomela
Wojciech Szpankowski
Suguru Tamaki
Seiichiro Tani
Sébastien Tavenas
Sharma V. Thankachan
Antonis Thomas
Himanshu Tyagi
Nikos Tzevelekos
Przemysław Uznański
Ali Vakilian
Pavel Valtr
Antonios Varvitsiotis
Yann Vaxès
Mikhail Volkov
Tjark Vredeveld
Stephan Wagner
Michael Walter
Pascal Weil
Oren Weimann
Stefan Weltge
Andreas Wiese
Richard Wilke
Ryan Williams
Anthony Wirth
Carsten Witt
Gerhard J. Woeginger
Burkhart Wolff
Marcin Wrochna
Amir Yehudayoff
Peter Zeman
Georg Zetzsche
Krzysztof Ziemiański
Martin Zimmermann
Anke van Zuylen

Recursive Combinatorial Structures: Enumeration, Probabilistic Analysis and Random Generation

Bruno Salvy
Inria, LIP (U. Lyon, CNRS, ENS Lyon, UCBL), France
Bruno.Salvy@inria.fr

Abstract

In a probabilistic context, the main data structures of computer science are viewed as random combinatorial objects. Analytic Combinatorics, as described in the book by Flajolet & Sedgewick, provides a set of high-level tools for their probabilistic analysis. Recursive combinatorial definitions lead to generating function equations from which efficient algorithms can be designed for enumeration, random generation and, to some extent, asymptotic analysis. With a focus on random generation, this tutorial first covers the basics of Analytic Combinatorics and then describes the idea of Boltzmann sampling and its realisation.

The tutorial addresses a broad TCS audience and no particular pre-knowledge on analytic combinatorics is expected.

2012 ACM Subject Classification Mathematics of computing → Generating functions, Theory of computation → Generating random combinatorial structures

Keywords and phrases Analytic Combinatorics, Generating Functions, Random Generation

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.1

Category Tutorial

Funding This work was partially supported by FastRelax ANR-14-CE25-0018-01.

1 Introduction

Combinatorial objects defined recursively by simple local rules tend to behave fairly regularly at large sizes. For instance, binary trees are defined by having nodes that are either leaves or binary internal nodes. From there, it turns out that all large random binary trees "look" the same. Also, many of their asymptotic characteristic persist for other classes of trees. The goal of analytic combinatorics is to understand and quantify those types of phenomena. Typical questions for binary trees could be: for a binary tree drawn uniformly at random among all binary trees with n nodes, what is the probability that the root has a leaf as one of its children? what is the expected distance from a random node to the root? what is the limiting distribution of this quantity?

The main applications are the probabilistic analysis of the average-case complexity of data structures and algorithms. Besides general "universality laws" of random discrete structures, the theory leads to very precise quantitative results. Analytic combinatorics is an active field of research, whose central core is described in detail, with numerous interesting examples, in the reference book *Analytic Combinatorics* by Flajolet and Sedgewick, published in 2009 and freely (and legally) available on-line [6]. Many methods of this theory can be made effective; this will be the focus of the tutorial.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 1; pp. 1:1–1:5

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


2 Constructible Classes

The classes of structures to which Analytic Combinatorics applies most directly are called *constructible*. They form an extension of context-free languages and can be defined recursively from a finite number of letters called atoms, of size 1 or 0, and combinatorial combinators: cartesian product, disjoint union, sequences, cycles and sets.

For example, the class of binary trees will be defined by the equation $\mathcal{B} = 1 + \mathcal{Z} \times \mathcal{B} \times \mathcal{B}$, expressing that a binary tree is either a '1' (a leaf encoded as an atom of size 0) or a triple formed by a root ($\mathcal{Z}$, an atom of size 1) and two binary trees attached to it. In terms of this class, the class of binary trees whose root has a leaf-child will be expressed by $\mathcal{Z} \times \mathcal{B} \times 1 + \mathcal{Z} \times 1 \times \mathcal{Z}$ (except that the binary tree with only one node is counted twice, which will not matter here). The class of non-planar rooted trees, also called Cayley trees, will be defined by $\mathcal{T} = \mathcal{Z} \times \mathrm{Set}(\mathcal{T})$, expressing that a tree is a node to which an arbitrary number of trees in an arbitrary order are attached. Series-parallel graphs will be defined by a system $\{\mathcal{G} = \mathcal{Z} + \mathcal{S} + \mathcal{P}, \mathcal{S} = \mathrm{Seq}_{>0}(\mathcal{Z} + \mathcal{P}), \mathcal{P} = \mathrm{Set}_{>0}(\mathcal{Z} + \mathcal{S})\}$, expressing that a graph ($\mathcal{G}$) is either a node ($\mathcal{Z}$) or a series graph ($\mathcal{S}$) or a parallel graph ($\mathcal{P}$), defined recursively in terms of each other. Such a system will be called a *specification* for the class. The motto of the theory is that "if a class can be specified, it can be analyzed."

3 Enumeration and Generating Functions

Given such a combinatorial specification for a class $\mathcal{F}$, the enumeration problem is to count the number f_n of distinct objects of a given size n in $\mathcal{F}$. For instance, knowing the number C_n (the Catalan number) of binary trees of size n and the number of those whose root has a leaf as one of its children lets one compute the probability that this event occurs.

The enumeration is simplified by the use of *generating functions*. The ordinary generating function of a sequence (f_n) is the formal power series $F(z) = \sum_{n\geq 0} f_n z^n$, while the exponential generating function is $\sum_{n\geq 0} f_n z^n/n!$. The use of one or the other depends on what exactly is counted as different. For instance there is only 1 sequence (list) of n atoms of size 1 if they are all identical, but $n!$ such sequences if they all carry a different label from 1 to n. Thus one distinguishes two 'universes': in the unlabelled universe, all atoms are alike and one uses ordinary generating functions, while in the labelled universe, an object of size n is formed of atoms labelled from 1 to n and one uses exponential generating functions. In both cases, there is an explicit dictionary translating the combinatorial specification into a system of equations for generating functions. Atoms of size 1 are translated into z and those of size 0 into $1 = z^0$; disjoint unions ('+') become additions and cartesian products become products of series. A sequence $\mathrm{Seq}(\mathcal{A})$ is translated into $1/(1 - A(z))$, where $A(z)$ is the generating function of $\mathcal{A}$. These first translation rules are identical in the labelled and unlabelled universes. Thus, in the example of binary trees, the equation is $B(z) = 1 + zB(z)^2$ in both cases and has for only power series solution the generating function of the Catalan numbers C_n. Similarly, in terms of this generating function, binary trees with one leaf-child at the root have generating function $2zB(z) - z$ (the term $-z$ discards the spurious tree of size 1), so that the probability mentioned in the introduction is $2C_{n-1}/C_n$ for $n > 1$. For sets and cycles, the rules differ, but remain simple in the labelled case, where Set becomes exp and $\mathrm{Cycle}(\mathcal{A})$ becomes $\log 1/(1 - A(z))$. For instance, in the case of Cayley trees, the exponential generating function is thus seen to satisfy $T(z) = z \exp(T(z))$.

4 Newton Iteration and Fast Enumeration

These equations give a first means to compute the enumeration sequences since they are fixed point equations over formal power series. A much more efficient algorithm is obtained by using Newton iteration for power series. The classical numerical Newton iteration solves an equation $f(y) = 0$ by approaching its root using the solution of successive linearized equations. Each iterate $y_{n+1} = y_n - f(y_n)/f'(y_n)$ is closer to the root when y_0 has been chosen appropriately. The extension of Newton iteration to formal power series is a standard algorithm in computer algebra [9]. Thus, for binary trees the Newton iteration reads $B_{n+1} = B_n + (1 + zB_n^2 - B_n)/(1 - 2zB_n)$. With the choice $B_0 = 0$, it produces a sequence of power series satisfying $B(z) - B_n = O(z^{2^n-1})$. Together with fast Fourier transform (FFT), Newton iteration lets one enumerate all constructible classes in quasi-optimal complexity [8][1]. It is even possible to obtain a combinatorial interpretation of the coefficients of the intermediate power series computed during the iteration: they enumerate combinatorial classes defined by a further lifting of the Newton iteration to combinatorial equations themselves, in the context of species theory [2, 1]. (Roughly speaking, species theory provides a sound theoretical basis grounded in category theory for what we have called "combinatorial class" so far.) For instance, in the case of binary trees, the combinatorial iteration then reads $\mathcal{B}_{n+1} = \mathcal{B}_n + \mathrm{Seq}(\mathcal{Z} \times \mathcal{B}_n \times \star + \mathcal{Z} \times \star \times \mathcal{B}_n) \times (1 + \mathcal{Z} \times \mathcal{B}_n \times \mathcal{B}_n \setminus \mathcal{B}_n)$, where $\star$ denotes an atom of size 0, which is interpreted as a 'bud' where trees can grow. The generating series of $\mathcal{B}_n$ is exactly the power series B_n produced by Newton iteration over power series.

5 Random Generation

Producing large random objects is the basis for simulation in a discrete world. Typical applications are the empirical evaluation of various parameters, software testing and the refinement of combinatorial models to suit an application.

A new family of efficient random generators called *Boltzmann samplers* was discovered at the beginning of the century [3, 4]. The principle is to draw each object of size n in a class $\mathcal{T}$ with a probability proportional to x^n for some prescribed positive real x and a factor of proportionality chosen so that the sum of probabilities is 1. Since the sum over all $t \in \mathcal{T}$ of $x^{\mathrm{size}(t)}$ is nothing but the evaluation of the generating function T of $\mathcal{T}$ at x, the probability will be $x^n/T(x)$, provided x is at most the radius of convergence of T. The algorithm of Boltzmann sampling itself is extremely simple and fits in about 10 lines. For atoms, the generator returns the atom; for cartesian products, it simply calls itself recursively on each of the components and assembles the results; for a disjoint union $\mathcal{A} + \mathcal{B}$, a random real number $t \in [0, 1]$ is compared to $u = A(x)/(A(x) + B(x))$ and the generator is called recursively on $\mathcal{A}$ if $t < u$ and on $\mathcal{B}$ otherwise. It is a simple exercise to check that the probabilities work out as expected. The values like $A(x)$ are provided by a numerical Newton iteration initialized at 0 [8]. The value of x can be adjusted to target a specific expected size $xT'(x)$ of the generated object.

[1] This is implemented in the `NewtonGF` Maple package, available at `http://perso.ens-lyon.fr/bruno.salvy/software/the-newtongf-package/`.

6 Asymptotic Analysis

Generating functions are also an entry point for the asymptotic analysis of their coefficients. The principle is to take these generating functions defined initially as power series given by a sequence of coefficients and view them as analytic functions in the complex plane. If $A(z) = \sum_{n\geq 0} a_n z^n$ has a positive radius of convergence R, then by Cauchy's theorem, its coefficient a_n can be recovered by the integral $a_n = \frac{1}{2\pi i} \oint (A(z)/z^{n+1})\, dz$, where for the contour of integration one can take a small circle around the origin of radius smaller than R. Classical complex analysis deforming the contour to locations where the integral concentrates asymptotically leads to very explicit and general results known as *transfer theorems*. The most useful case is when there is a unique singularity on the circle of convergence, at a point ρ (which will always be real), where the generating function behaves like $c(1 - z/\rho)^\alpha$, for $\alpha \notin \mathbb{N}$. Then the coefficients behave asymptotically like $c\rho^{-n} n^{-\alpha-1}/\Gamma(-\alpha)$. The outcome is a general 3-step method for the asymptotic analysis of generating functions: *(i).* locate the singularities of minimal modulus; *(ii).* compute the local behaviour of the generating function there (which can often be found directly from the defining equations); *(iii).* translate using a very simple dictionary. This process can be automated in a large part and full asymptotic expansions computed using computer algebra systems. In the case of binary trees, the quadratic equation satisfied by the generating function can be solved explicitly to yield $B(z) = (1 - \sqrt{1-4z})/(2z)$. From there, the explicit formula for the Catalan numbers $C_n = \binom{2n}{n}/(n+1)$ can be derived and the asymptotic behaviour could be computed by Stirling's formula. However, such closed forms are the exception rather than the rule, and the method applies in general. There, the dominant singularity is $1/4$, the local behaviour is $2 - 2\sqrt{1-4z} + O(1-4z)$. The coefficients of $B(z) - 2$ which are C_n for $n > 0$ thus behave asymptotically like $-2 \cdot 4^n n^{-3/2}/\Gamma(-1/2) = 4^n n^{-3/2}/\sqrt{\pi}$. One can also deal with the probability of a leaf-child at the root either using the formula $2C_{n-1}/C_n$ from §3 and simplification of binomials, or by applying the general method to the generating function $2zB(z) - z$. The singularity is the same as that of $B(z)$; the local behaviour at first order is $2\rho = 1/2$ times that of $B(z)$, so that $1/2$ is the limiting probability. A slighly more detailed computation by this method yields $1/2 + 3/(4n) + O(1/n^2)$.

These asymptotic techniques help analysis of parameters of combinatorial structures. One then introduces *multivariate* generating functions of the form $F(z,u) = \sum_{n,k} f_{n,k} u^k z^n$ (and the labelled counterpart), where $f_{n,k}$ denotes the number of objects of size n for which the parameter of interest takes the value k. As a concrete example, one can think of the internal path-length in a binary tree, that is, the sum of the distances from the nodes to the root. Equations for the generating series can often be derived by an extended dictionary [7]. In the case of path-length in binary trees, the equation reads $B(z,u) = z + B^2(zu,u)$. From such a bivariate generating function, the expected value of the parameter on objects of size n is obtained by dividing the coefficient of z^n in $\partial F/\partial u|_{u=1}$ by that of z^n in F. Both are univariate generating functions to which the previous method applies. In our example, one gets an average distance of the nodes to the roots growing like $\sqrt{\pi n}$.

The next step of Analytic Combinatorics is the study of limiting distributions of parameters, e.g., path-length in binary trees is asymptotically Gaussian after proper normalization. This goes beyond what can be covered in this tutorial and we refer the reader to the last part of the book by Flajolet & Sedgewick.

7 Conclusion

The message of this tutorial is that from a combinatorial specification, Analytic Combinatorics provides easy-to-use tools that provide counting, random generation and asymptotic analysis. Work is under way to automate this approach fully within computer algebra. Counting and the required parts of random generation are complete and asymptotic analysis is only partly done, but progress is being made. The ultimate goal would be a system taking as input a combinatorial specification and some sort of description of an algorithm and producing automatically the asymptotic average-case behaviour of the algorithm. The approach was tested a long time ago and works well for various grammars and parameters [5], but much remains to be done.

References

1 F. Bergeron, G. Labelle, and P. Leroux. *Combinatorial species and tree-like structures*. Number 67 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge, 1998. Translated from the 1994 French original by Margaret Readdy, With a foreword by Gian-Carlo Rota.

2 H. Décoste, G. Labelle, and P. Leroux. Une approche combinatoire pour l'itération de Newton-Raphson. *Advances in Applied Mathematics*, 3(4):407–416, 1982.

3 Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability and Computing*, 13(4–5):577–625, 2004. Special issue on Analysis of Algorithms.

4 Philippe Flajolet, Éric Fusy, and Carine Pivoteau. Boltzmann sampling of unlabelled structures. In David Applegate, Gerth Stølting Brodal, Daniel Panario, and Robert Sedgewick, editors, *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithmics and Combinatorics*, volume 126 of *SIAM Proceedings in Applied Mathematics*, pages 201–211. SIAM, 2007. Workshops held in New Orleans, LA, January 6, 2007.

5 Philippe Flajolet, Bruno Salvy, and Paul Zimmermann. Automatic average-case analysis of algorithm. *Theor. Comput. Sci.*, 79(1):37–109, 1991. doi:10.1016/0304-3975(91)90145-R.

6 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009. 824 pages (ISBN-13: 9780521898065); also available electronically from the authors' home pages.

7 Marni Mishna. Attribute grammars and automatic complexity analysis. *Advances in Applied Mathematics*, 30(1):189–207, 2003.

8 Carine Pivoteau, Bruno Salvy, and Michèle Soria. Algorithms for combinatorial structures: Well-founded systems and newton iterations. *J. Comb. Theory, Ser. A*, 119(8):1711–1773, 2012. doi:10.1016/j.jcta.2012.05.007.

9 Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge University Press, New York, 2nd edition, 2003. URL: http://www.cambridge.org/fr/knowledge/isbn/item1170826.

Lower Bound Techniques for QBF Proof Systems

Meena Mahajan
The Institute of Mathematical Sciences, HBNI, Chennai, India
meena@imsc.res.in

Abstract

How do we prove that a false QBF is indeed false? How big a proof is needed? The special case when all quantifiers are existential is the well-studied setting of propositional proof complexity. Expectedly, universal quantifiers change the game significantly. Several proof systems have been designed in the last couple of decades to handle QBFs. Lower bound paradigms from propositional proof complexity cannot always be extended - in most cases feasible interpolation and consequent transfer of circuit lower bounds works, but obtaining lower bounds on size by providing lower bounds on width fails dramatically. A new paradigm with no analogue in the propositional world has emerged in the form of strategy extraction, allowing for transfer of circuit lower bounds, as well as obtaining independent genuine QBF lower bounds based on a semantic cost measure.

This talk will provide a broad overview of some of these developments.

2012 ACM Subject Classification Theory of computation → Proof complexity, Theory of computation → Complexity theory and logic

Keywords and phrases Proof Complexity, Quantified Boolean formulas, Resolution, Lower Bound Techniques

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.2

Category Invited Talk

Funding This work was partially supported by the EU Marie Curie IRSES grant CORCON.

Acknowledgements I thank Olaf Beyersdorff, from whom I learnt all I know about QBF proof complexity, and Anil Shukla, who helped immensely by learning along with me.

1 Introduction

Despite the NP-completeness of SAT, SAT solvers have proven to be highly successful in tackling humongous instances of satisfiability arising in practical applications. This has spurred more ambitious programs to develop practical solvers for more complex and expressive formulas. In the last couple of decades, several solvers have been developed to decide the truth or falsity of Quantified Boolean Formulas QBFs, a PSPACE-complete problem. As in the case of SAT, underlying the solvers are proof systems – formal systems where the truth or falsity of a QBF is established through a sequence of easily checkable steps. A natural measure of efficiency is the number of steps in such a proof, since it corresponds to the length of the run of the solver. Understanding the limitations of a solver is thus intimately connected to understanding the limitations of a proof system; hence the quest for explicit lower bounds in proof systems.

2 Proof systems for QBF

What does a typical proof system for QBFs look like? One could start with the standard propositional proof systems, where one proves that a formula is not satisfiable (that is, a QBF

with only existentially quantified variables is false), and strengthen it to handle universally quantified variables. An obvious starting point is the well-studied resolution system Res, that works with unsatisfiable formulas in conjunctive normal form CNF. If a set $\mathcal{C}$ of clauses is simultaneously satisfiable by an assignment $\tilde{a}$, and if $\mathcal{C}$ contains clauses $A \vee x$, $B \vee \neg x$, then the set $\mathcal{C}' = \mathcal{C} \cup \{A \vee B\}$ is also satisfied by $\tilde{a}$. Using this resolution rule, in which we call x the pivot variable and $A \vee B$ the resolvent, we repeatedly enlarge the set of clauses until it contains the empty clause $\Box$; this set of clauses is patently unsatisfiable. This allows us to conclude that the original set of clauses must also have been unsatisfiable.

To strengthen this system to handle quantifications, it is useful to consider the evaluation game played on QBFs. We assume that the QBF is in prenex CNF: all variables are quantified first, and then there is a matrix of clauses. There are two players. The existential player owns the existentially quantified variables, the universal player owns the rest. The players step through the quantifier prefix, and as each variable is encountered, the player who owns it declares a value for it. The existential player wins a run of the game if the constructed assignment satisfies all clauses in the matrix; otherwise the universal player wins. A QBF is true if and only if the existential player has a winning strategy that allows him to win no matter how the universal player plays; it is false if and only if the universal player has a winning strategy. Thus, a strategy for the universal player, and a proof that it is a winning strategy, is a proof that the QBF is false.

We can now consider three different approaches to augmenting Res or other propositional proof systems to handle QBFs; more specifically, to handle universally quantified variables.

Eliminate-by-expansion

Remove the universal variables altogether! Use the semantics $\forall u F(u) \equiv F(0) \wedge F(1)$, but to avoid explosion of formula size, do the expansion on-the-fly, so to say. Since, in a run of the QBF game, values of existential variables can depend on those of the preceding universal variables, we make appropriate copies of existential variables, annotated with assignments to preceding universal variables. When using a clause from the matrix in the proof, the universal variables in the clause must be set to false. Other universal variables need not be set at this stage. That is, the annotations can be complete or partial. Now use standard resolution, keeping in mind that a single existential variable with two different annotations must be treated as two different variables.

The systems ∀Exp+Res (∀Expansion + Resolution), IR (Instantiation + Resolution) are based on this idea; see [22] and [8]. The solvers CAQE [27], CEGAR [18], Ghost-Q [24], RAReQS [21] use such expansion-based ideas.

Eliminate-via-∀-reduction

Consider an intermediate stage during a run of the game on a QBF. If the partial assignment constructed so far results in a clause getting simplified to one with only un-assigned universal variables, then the universal player can win by simply falsifying this clause. This gives rise to the ∀-reduction rule that can be added on to any line-based propositional system. The simplest such system, and indeed, one of the earliest formal proof systems for QBFs, is the system Q-Res, see [23], where resolution can be performed on existential pivots, tautologies must be discarded, and a clause A can be inferred from a clause $A \vee u$ where u is a universal literal and all variables in A appear left of u in the prefix. A natural generalisation is QU-Res, where the pivot for resolution can also be universal, see [30]. A more informative name for QU-Res is perhaps Res + ∀Red (Resolution + ∀Reduction). Many DPLL-based solvers

using conflict-driven-clause-learning, eg Evaluate, [15, 16], QuBe, [20] are based on such systems. In a similar vein, one could add a ∀ Reduction rule to Frege proof systems or to the Polynomial Calculus system and obtain proof systems sound and complete for QBFs; see [7].

Merging complementary literals

When performing resolution, tautologies are removed because they contribute nothing to proofs of unsatisfiability. In the case of QBFs, however, what looks like a tautology may not really be one. If a resolution rule produces a clause containing u and $\neg u$ for some universal variable u, such a clause could be still be useful, because the two complementary literals come from different sub-derivations, and a winning strategy could use this information to decide how to set u. So instead of discarding the clause, we retain a version of it, replacing the literals $u, \neg u$ with a single merged literal u^*. This is referred to as long-distance resolution. To preserve soundness of the rule, some side-conditions are imposed on when such a resolution and merging is permissible.

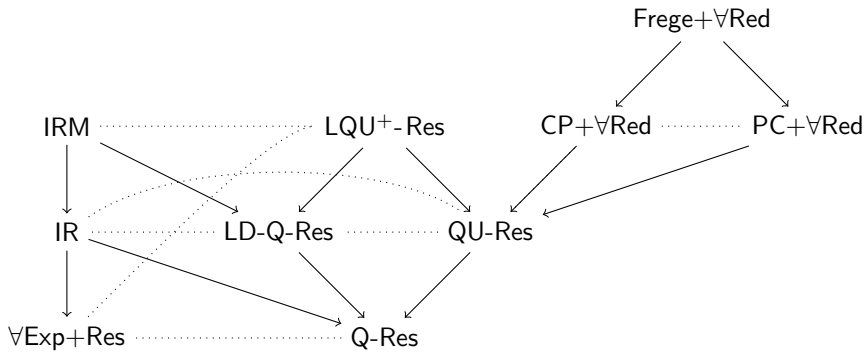
Augmenting Q-Res and QU-Res with long-distance resolution gives rise to the proof systems LD-Q-Res and LQU$^+$-Res; see [2, 3].

A similar situation can also arise in the expansion-based systems, where a variable appears in the resolvent with annotations that differ in their assignment to universal variables. Again, under appropriate side-conditions, a merger of the annotations is permissible. This gives rise to the proof system IRM (Instantiation, Resolution, Merge); see [8].

Many solvers use long-distance resolution very effectively; for instance, Quaffle [31].

The relative power of some QBF proof systems

The figure below shows some of the relationships between the QBF proof systems discussed. An arrow from A to B indicates that a proof in system B can be transformed (in polynomial time) to a proof in system A; that is, system A p-simulates system B. A dotted line indicates that neither p-simulates the other; they are incomparable.



3 Where the hardness comes from

3.1 Propositional hardness

An unsatisfiable CNF formula is a false QBF with only existential variables. A proof of its falsity in a QBF proof system is just a proof of unsatisfiability. If this is hard to demonstrate in the specialisation of the QBF proof system to purely existential formulas, then of course it is a hard QBF for the QBF proof system. Thus, for instance, the pigeonhole principle

formula, asserting that $m+1$ pigeons can be placed in m holes without collision, is known to require exponentially long proofs in Res; trivially, it then requires exponentially long proofs in Q-Res and ∀Exp+Res. The Clique-color formulas, asserting that there is a k-colorable graph with a clique of size $k+1$, require exponentially long proofs in Cutting Planes; hence they require long proofs in CP+∀Red as well.

While propositional hardness is indeed a valid source of hardness for QBF proof systems, it does not tell us anything new about the ability (or lack thereof) of a QBF solver to handle universal variables. Thus in QBF proof complexity, such hardness is not particularly interesting.

3.2 Adapting techniques from propositional hardness

While pure propositional hardness may not be interesting, it is reasonable to expect that techniques used to establish such hardness could perhaps be adapted to prove non-trivial QBF hardness.

The size-width technique fails. The central lower-bound technique in the case of resolution is the relation between the size of proofs and their width (the maxmimum number of literals in any clause in the proof), due to [4]. The width of proofs also yields lower bounds on the space complexity; see [1]. Unfortunately, this technique fails completely in the case of the simplest extension, Q-Res, as shown in [10]; there are formulas with short proofs, derivable using very little space, but the width of any proof for these formulas must be large.

Feasible interpolation works. The technique of feasible interpolation exploits known (monotone) circuit lower bounds to obtain lower bounds for proofs of formulas of a specific type. It was used to show exponential lower bounds in the propositional proof systems Res and Cutting Planes, see [25, 26]. The technique can be adapted to similarly obtain lower bounds in QBF proof systems as well. The set-up is as follows: We start with a false QBF of the form

$$\varphi = \exists \vec{p}\ \mathcal{Q}\vec{q}\ \mathcal{Q}\vec{r}\ \cdot\ [A(\vec{p},\vec{q}) \wedge B(\vec{p},\vec{r})].$$

For every assignment $\vec{a}$ to the common variables $\vec{p}$, either $\mathcal{Q}\vec{q}\cdot A(\vec{a},\vec{q})$ or $\mathcal{Q}\vec{r}\cdot B(\vec{a},\vec{r})$ (or both) must be false. From a proof π that φ is false, we extract a circuit C in the $\vec{p}$ variables with the property that $C(\vec{a})=0$ implies $\mathcal{Q}\vec{q}\cdot A(\vec{a},\vec{q})$ is false and $C(\vec{a})=1$ implies $\mathcal{Q}\vec{r}\cdot B(\vec{a},\vec{r})$ is false. That is, C computes an interpolant. Furthermore, the size of C is polynomial in the size of π. Now, if interpolants for a formula are known to require large circuits, it follows that the formula cannot have small proofs. If the extracted circuit is monotone, then monotone circuit hardness gives a proof size lower bound.

In [11, 12] it is shown that all the Res-based QBF proof systems (Q-Res, QU-Res, LD-Q-Res, LQU$^+$-Res, ∀Exp+Res, IR, IRM), as well as the proof system CP+∀Red, admit monotone feasible interpolation. Hence the Clique-co-Clique formulas, asserting that a graph both has and does not have a large clique, are hard to prove false in these systems.

3.3 Winning strategies hard for decision lists

The universal player has a winning strategy in the evaluation game played on a false QBF. In general, a winning strategy may not be easy to compute. However, proofs in most proof systems reveal strategies. Let π be a proof of falsity in some QBF proof system of the form $P+\forall\text{Red}$, where P is some propositional proof system. By examining π, the player can

figure out a way to compute a winning strategy. The computation focuses on the ∀ Reduction steps in the proof. Let $L_1, L_2, \ldots, L_m$ be the lines of the proof, with the last line being something obviously false (such as the empty clause □). Suppose it is the universal player's turn to play, and she has to choose a value for the variable u. All variables left of u in the prefix have already been assigned, and she knows the partial assignment $\vec{a}$. She now looks at the lines in the proof that are obtained by a ∀ reduction applied to the variable u. Let the indices of these lines be $(1 <) i_1 < \ldots < i_k (\leq m)$, and let each L_{i_r} be obtained from some L_{j_r}, $j_r < i_r$, by dropping u or $\neg u$. Note that since dropping u was permitted, each L_{i_r} is either true or false under $\vec{a}$; there are no variables still unassigned. She steps through the following decision list:

if	$L_{i_1}(\vec{a}) = 0$	**then**	set u to make $L_{j_1}(\vec{a}) = 0$
elseif	$L_{i_2}(\vec{a}) = 0$	**then**	set u to make $L_{j_2}(\vec{a}) = 0$
⋮			
elseif	$L_{i_k}(\vec{a}) = 0$	**then**	set u to make $L_{j_k}(\vec{a}) = 0$
else			set $u = 0$.

This can be shown to be a winning strategy; see [9, 7]. Note that the kind of decisions to be made while computing it depend on the nature of the lines allowed in the proof system $P + \forall\text{Red}$.

Now, if the formula has the property that there is a unique winning strategy for some variable, and if this winning strategy function is known to require large decision lists of the type specified by the proof system, then it follows that the proof must have large size. What is more, it must be large not because of the steps from the propositional part P (although that too may be the case), but because it has many ∀Reduction steps. This may be the case even if there are very few universal variables. A simple example is the formula QParity: $\exists \vec{x}\ \forall z\ \exists \vec{t}\ A(\vec{x}, \vec{t}) \wedge (t_n \vee z) \wedge (\neg t_n \vee \neg z)$, where the clauses in A express the property that t_n computes the parity of $x_1, \ldots, x_n$. (for example, clauses equivalent to $t_2 = x_1 \oplus x_2$, $t_i = t_{i-1} \oplus x_i$ for $i > 2$). The only winning strategy for the universal player is to choose z to be the parity of $x_1, \ldots, x_n$. A Q-Res or QU-Res proof with S lines would give a decision list computing parity with at most S decision steps, where each decision involves evaluating a clause on an assignment. If S were polynomial in n, this would give a way of computing parity in AC^0, something we know is not possible. So this formula has no short proof in Q-Res or QU-Res ([9]). Similarly, a formula asserting that for some $\vec{x}, \vec{y}$, the inner product modulo 2 is both 0 and 1 is hard for CP+∀Red ([12]) because the inner product function requires exponentially long decision lists of threshold functions ([29]).

See [9, 7, 12] for more examples of such hardness. In particular, it is noteworthy that this technique gives explicit lower bounds against proof systems $\text{AC}^0[p]$-Frege+∀Red for any prime p. In contrast, in the propositional world, the strongest lower bound holds for the system AC^0-Frege, while lower bounds for $\text{AC}^0[p]$-Frege remains an outstanding open question.

The feasible interpolation technique for QBFs described earlier is essentially a special case of strategy extraction; see [11].

3.4 Winning strategies requiring varied responses

In [14], it is shown that hardness in Frege+∀Red must stem from either propositional hardness in the system Frege or from a circuit lower bound. (Thus any hardness proof for Frege+∀Red would constitute a major advance, either in proof complexity or in circuit complexity.) There are no other sources of hardness. This is not the case for weaker systems. Consider the

formula QEquality:

$$\exists x_1 \cdots x_n \forall u_1 \cdots u_n \exists t_1 \cdots t_n \left(\bigwedge_{i=1}^{n} (x_i \vee u_i \vee t_i) \wedge (\neg x_i \vee \neg u_i \vee t_i) \right) \wedge \left(\bigvee_{i=1}^{n} \neg t_i \right)$$

It has a unique winning strategy that is extremely simple to compute: just let $u_i = x_i$ for each $i \in [n]$. Thus the decision list approach cannot show that this formula is hard in any system. However, it turns out to require exponentially long proofs in QU-Res. The reason is as follows: the winning strategy has a range of size 2^n; all responses are required. However, the algorithm that extracts a winning strategy from a proof builds up responses line-by-line, with each line contributing a limited amount to the range. In particular, in QU-Res, each line is a clause and contributes a single response. It follows that there must be many lines.

Generalising this idea needs some work and yields the very elegant size-cost-capacity theorem, see [6], applicable to proof systems of the form $P + \forall\text{Red}$. The cost of a formula is the number of distinct responses required in any winning strategy. A caveat: it is important here that we only count responses to universal variables in a single quantification block (which block doesn't matter). The capacity of a proof is an upper bound on the number of responses that a single line in a proof can contribute. The theorem says that the cost of a formula is bounded above by the size of a proof times the capacity of the proof.

In the case of QU-Res and CP+∀Red, the capacity of any proof is simply 1, and thus the formula cost is itself a proof size lower bound. In the case of PC+∀Red, the capacity of a proof is no more than its size, and so proof size is at least square root of the formula cost.

The size-cost-capacity theorem has bene used to show families of random QBFs hard in QU-Res, CP+∀Red and PC+∀Red.

Paralleling the size-cost-capacity theorem, while dealing with expansion-based systems, strategy size and weight provide proof size lower bounds; these results are established by counting annotations, and are reported in [5].

A prominent example of hard QBFs are the KBKF formulas introduced in [23]. These are known to require large proofs in Q-Res and IR [23, 9], but have short proofs in QU-Res [30] and in LD-Q-Res [19]. The proofs of hardness are length and cumbresome, and essentially use an ad hoc combinatorial argument. The size-cost-capacity technique provides an alternative, and, arguably, more insightful, proof that the KBKF formulas require exponentially long proofs in Q-Res and that a doubled variant requires exponentially long proofs in QU-Res, CP+∀Red, PC+∀Red. Similarly, the strategy-size theorem provides a more insightful proof that the KBKF formulas require exponentially long proofs in IR.

4 Conclusion

QBF proof complexity is a relatively young field, but it has already thrown up very interesting insights and techniques. A good starting point to read about QBF proof complexity are the doctoral dissertations of Leroy Chew [17] and Anil Shukla [28], although there have already been quite a few advances after that, eg [13, 6, 5]. In this short article, I have not tried to be exhaustive, and I apologise to the readers who find their favourite papers missing.

References

1 Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *J. Comput. Syst. Sci.*, 74(3):323–334, 2008. URL: https://doi.org/10.1016/j.jcss.2007.06.025, doi:10.1016/j.jcss.2007.06.025.

2 Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Formal Methods in System Design*, 41(1):45–65, 2012.

3 Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In *Proceeding of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 154–169, 2014.

4 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow - resolution made simple. *Journal of the ACM*, 48(2):149–169, 2001.

5 Olaf Beyersdorff and Joshua Blinkhorn. Genuine lower bounds for QBF expansion. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science (STACS)*, page to appear, 2018.

6 Olaf Beyersdorff, Joshua Blinkhorn, and Luke Hinde. Size, cost, and capacity: A semantic technique for hard random QBFs. In *Proceedings of the ACM Conference on Innovations in Theoretical Computer Science (ITCS)*, page to appear, 2018.

7 Olaf Beyersdorff, Ilario Bonacina, and Leroy Chew. Lower bounds: From circuits to QBF proof systems. In *Proceedings of the ACM Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 249–260. ACM, 2016.

8 Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. On unification of QBF resolution-based calculi. In *Proceedings of 39th International Symposium on Mathematical Foundations of Computer Science MFCS, Proceedings Part II, LNCS 8635*, pages 81–93, 2014.

9 Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. Proof complexity of resolution-based QBF calculi. In *Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 76–89. LIPIcs, 2015.

10 Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla. Are short proofs narrow? QBF resolution is not so simple. *ACM Transactions on Computational Logic*, 19(1):1:1–1:26, Dec 2017. (preliminary version in STACS 2016).

11 Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla. Feasible Interpolation for QBF Resolution Calculi. *Logical Methods in Computer Science*, Volume 13, Issue 2, 2017. (preliminary version in ICALP 2015).

12 Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla. Understanding cutting planes for QBFs. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:37, 2017. (preliminary version in FSTTCS 2016).

13 Olaf Beyersdorff, Luke Hinde, and Ján Pich. Reasons for hardness in QBF proof systems. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016)*, pages 14:1–14:14, 2017.

14 Olaf Beyersdorff and Ján Pich. Understanding Gentzen and Frege systems for QBF. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 146–155, 2016.

15 Marco Cadoli, Andrea Giovanardi, and Marco Schaerf. An algorithm to evaluate Quantified Boolean Formulae. In *Proceedings of the 15th National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference (AAAI)*, pages 262–267, 1998.

16 Marco Cadoli, Marco Schaerf, Andrea Giovanardi, and Massimo Giovanardi. An algorithm to evaluate Quantified Boolean Formulae and its experimental evaluation. *Journal of Automated Reasoning*, 28(2):101–142, 2002.

17 Leroy Chew. *QBF Proof Complexity*. PhD thesis, University of Leeds, UK, 2017.

18 Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003.

19 Uwe Egly, Florian Lonsing, and Magdalena Widl. Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In *19th International Conference*

on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR), pages 291–308. Springer Berlin Heidelberg, 2013.

20 Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. QuBE++: An efficient QBF solver. In *Formal Methods in Computer-Aided Design*, pages 201–213. Springer Berlin Heidelberg, 2004.

21 Mikolás Janota, William Klieber, João Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. In *Proceeding of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 114–128, 2012.

22 Mikolás Janota and Joao Marques-Silva. Expansion-based QBF solving versus Q-resolution. *Theoretical Computer Science*, 577:25–42, 2015.

23 Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified Boolean formulas. *Information and Computation*, 117(1):12–18, 1995.

24 William Klieber, Samir Sapra, Sicun Gao, and Edmund M. Clarke. A non-prenex, non-clausal QBF solver with game-state learning. In *13th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 128–142, 2010.

25 Jan Krajíček. Interpolation theorems, lower bounds for proof systems and independence results for bounded arithmetic. *The Journal of Symbolic Logic*, 62(2):457–486, 1997.

26 Pavel Pudlák. Lower bounds for resolution and cutting planes proofs and monotone computations. *The Journal of Symbolic Logic*, 62(3):981–998, 1997.

27 Markus N. Rabe and Leander Tentrup. CAQE: A certifying QBF solver. In *Formal Methods in Computer-Aided Design, (FMCAD)*, pages 136–143, 2015.

28 Anil Shukla. *On Proof Complexity for Quantified Boolean Formulas.* PhD thesis, The Institute of Mathematical Sciences, HBNI, India, 2017.

29 György Turán and Farrokh Vatan. Linear decision lists and partitioning algorithms for the construction of neural networks. In *Foundations of Computational Mathematics*. Springer, 1997.

30 Allen Van Gelder. Contributions to the theory of practical quantified Boolean formula solving. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 647–663, 2012.

31 Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified Boolean satisfiability solver. In *International Conference on Computer Aided Design*, pages 442–449, 2002.

On the Positive Calculus of Relations with Transitive Closure

Damien Pous
Univ. Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP
Damien.Pous@ens-lyon.fr

Abstract

Binary relations are such a basic object that they appear in many places in mathematics and computer science. For instance, when dealing with graphs, program semantics, or termination guarantees, binary relations are always used at some point.

In this survey, we focus on the relations themselves, and we consider algebraic and algorithmic questions. On the algebraic side, we want to understand and characterise the laws governing the behaviour of the following standard operations on relations: union, intersection, composition, converse, and reflexive-transitive closure. On the algorithmic side, we look for decision procedures for equality or inequality of relations.

After having formally defined the calculus of relations, we recall the existing results about two well-studied fragments of particular importance: Kleene algebras and allegories. Unifying those fragments yields a decidable theory whose axiomatisability remains an open problem.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory, Theory of computation → Equational logic and rewriting

Keywords and phrases Relation Algebra, Kleene Algebra, Allegories, Automata, Graphs

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.3

Category Invited Talk

Funding This work has been supported by the European Research Council (ERC) under the European Union's Horizon 2020 programme (CoVeCe, grant agreement No 678157), and by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

Acknowledgements I would like to thank Paul Brunet, with whom we obtained the few new results presented in this survey, as well as Arnaud Durand for his feedback on a preliminary version of this survey, which appeared in French as course notes for a research school (EJCIM'16) [20].

1 The calculus of relations

Given a set P, a *relation* on P is a set of pairs of elements from P. For instance, the usual order on natural numbers is a relation. In the sequel, relations are ranged over using letters R, S, their set is written $\mathcal{P}(P\times P)$, and we write $p\,R\,q$ for $\langle p,q\rangle \in R$.

The set of relations is equiped with a partial order, set-theoretic inclusion ($\subseteq$), and three binary operations: set-theoretic union, written $R+S$, set-theoretic intersection, written $R\cap S$, and relational composition:

$$R\cdot S \triangleq \{\langle p,q\rangle \mid \exists r\in P,\ p\,R\,r\ \wedge\ r\,S\,q\}\ .$$

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 3; pp. 3:1–3:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

It also contains three specific relations: the empty relation, written 0, the full relation, written $\top$, and the identity relation:

$$1 \triangleq \{\langle p, p\rangle \mid p \in P\} \ .$$

Lastly, one can consider three unary operations: set-theoretic complement, written R^c, converse (or transpose), R°, and reflexive-transitive closure, $R^\star$, defined as follows:

$$R^c \triangleq \{\langle p, q\rangle \mid \neg p\, R\, q\} \ ,$$
$$R^\circ \triangleq \{\langle p, q\rangle \mid q\, R\, p\} \ ,$$
$$R^\star \triangleq \{\langle p, q\rangle \mid \exists p_0, \ldots, p_n,\ p_0 = p \ \wedge\ p_n = q \ \wedge\ \forall i < n,\ p_i\, R\, p_{i+1}\} \ .$$

We restrict ourselves to this list of operations here, even though it is not exhaustive. These operations make it possibles to state many properties in a concise way, without mentioning the points related by the relations. Here are a few examples:

$1 \subseteq R$	R is reflexive: $\forall p \in P, p\, R\, p$
$R{\cdot}R \subseteq R$	R is transitive: $\forall pqr,\ p\, R\, r \wedge r\, R\, q \Rightarrow p\, R\, q$
$R{\cdot}R^\star \cap 1 = 0$	R is acyclic: $\forall p_0 \ldots p_n,\ n{>}0,\ (\forall i,\ p_i\, R\, p_{i+1}) \Rightarrow p_0 \neq p_n$
$R^\circ{\cdot}S \subseteq S{\cdot}R^\circ$	R and S commute: $\forall pqr,\ r\, R\, p \wedge r\, S\, q \Rightarrow \exists t,\ q\, R\, t \wedge p\, S\, t$

▶ **Exercise 1.** *To which standard notions from rewriting theory correspond the inequations* $R^\circ{\cdot}R \subseteq R^\star{\cdot}R^{\circ\star}$ *and* $R^{\circ\star}{\cdot}R^\star \subseteq R^\star{\cdot}R^{\circ\star}$ *?*

Moreover, these operations satisfy many laws. Some of these laws are extremely simple (for instance, composition is associative, $(R{\cdot}R'){\cdot}R'' = R{\cdot}(R'{\cdot}R'')$; the empty relation is absorbs composition, $R0 = 0 = 0R$; reflexive-transitive closures are transitive, $R^\star{\cdot}R^\star \subseteq R^\star$). Others are much more complicated and counter-intuitive.

▶ **Exercise 2.** *Amongst the following equations and inequations, which ones are universally true? In each case, give a counter-example or a detailed proof.*

$$1 \cap R \subseteq R{\cdot}R \cap R{\cdot}R{\cdot}R \tag{1}$$
$$(R + S)^\star = R^\star{\cdot}(S{\cdot}R^\star)^\star \tag{2}$$
$$(R + S)^\star = ((1 + R){\cdot}S)^\star \tag{3}$$
$$R{\cdot}(S \cap T) = R{\cdot}S \cap R{\cdot}T \tag{4}$$
$$R{\cdot}S \cap T \subseteq R{\cdot}(S \cap R^\circ{\cdot}T) \tag{5}$$
$$R{\cdot}S \cap T \subseteq (R \cap T{\cdot}S^\circ){\cdot}(S \cap R^\circ{\cdot}T) \tag{6}$$
$$(R \cap S{\cdot}\top){\cdot}T = R{\cdot}T \cap S{\cdot}\top \tag{7}$$

Two questions arise naturally:

1. is it possible to axiomatise the set of laws that are universally true, that is, to give a small number of elementary laws from which all valid laws follow?
2. is it possible to decide whether a law is valid or not?

When considering all the operations listed above, the answer is negative in both cases. Indeed, Monk proved that there cannot be a finite axiomatisation [16], and Tarski proved that the theory is actually undecidable [25, 24]. In both cases, reflexive-transitive closure is not necessary but the complement plays a crucial role. Thus we focus in the sequel on the *positive* fragments, where complement is excluded.

Now we setup the concepts and notation needed in the sequel.

Let Σ be a set, whose elements are denoted by letters a, b. *(Relational) expressions* are defined by the following grammar:

$$e, f, g ::= e + f \mid e \cap f \mid e \cdot f \mid e^\circ \mid e^\star \mid 0 \mid 1 \mid \top \mid a \qquad (a \in \Sigma) \ .$$

Given a set E and a function $\sigma : \Sigma \to \mathcal{P}(E \times E)$ mapping any letter from Σ to a relation on E, we define inductively the extension $\hat{\sigma}$ of σ to expressions:

$$\begin{array}{lll} \hat{\sigma}(e+f) \triangleq \hat{\sigma}(e) + \hat{\sigma}(f) & \hat{\sigma}(e^\circ) \triangleq \hat{\sigma}(e)^\circ & \hat{\sigma}(1) \triangleq 1 \\ \hat{\sigma}(e \cap f) \triangleq \hat{\sigma}(e) \cap \hat{\sigma}(f) & \hat{\sigma}(e^\star) \triangleq \hat{\sigma}(e)^\star & \hat{\sigma}(\top) \triangleq \top \\ \hat{\sigma}(e \cdot f) \triangleq \hat{\sigma}(e) \cdot \hat{\sigma}(f) & \hat{\sigma}(0) \triangleq 0 & \hat{\sigma}(a) \triangleq \sigma(a) \end{array}$$

Given two expressions e and f, an equation is *valid*, written $\vDash e = f$, if for all set E and for all function $\sigma : \Sigma \to \mathcal{P}(E \times E)$, we have $\hat{\sigma}(e) = \hat{\sigma}(f)$. Intuitively, an equation is valid if it is universally true in relations, if it holds whatever the relations we use to interpret its variables.

Similarly, an inequation is valid, written $\vDash e \subseteq f$, if $\hat{\sigma}(e) \subseteq \hat{\sigma}(f)$ for all set E and function $\sigma : \Sigma \to \mathcal{P}(E \times E)$. Characterising valid equations is equivalent to characterising valid inequations, as shown in the following exercise.

▶ **Exercise 3.** *Let e, f be two expressions. We have $\vDash e = f$ iff $\vDash e \subseteq f$ and $\vDash f \subseteq e$. Show that $\vDash e \subseteq f$ iff $\vDash e + f = f$ iff $\vDash e \cap f = e$.*

2 The ideal fragment: Kleene algebra

In this section we remove from the syntax the operations of intersection and converse, as well as the constant $\top$. In other words, we restrict to regular expressions:

$$e, f, g ::= e + f \mid e \cdot f \mid e^\star \mid 0 \mid 1 \mid a \qquad (a \in \Sigma) \ .$$

we shall see that with such a restriction, the validity of an equation is decidable, and more precisely, PSPACE-complete.

2.1 Decidability

Let letter u, v range over finite words over the alphabet Σ, let ϵ denote the empty word, and uv the concatenation of two words u and v. A *language* is a set of words. We define inductively a function $[\cdot]$ associating a language to each expression:

$$\begin{array}{ll} [e+f] \triangleq [e] \cup [f] & [0] \triangleq \emptyset \\ [e \cdot f] \triangleq \{uv \mid u \in [e], v \in [f]\} & [1] \triangleq \{\epsilon\} \\ [e^\star] \triangleq \{u_1 \dots u_n \mid \forall i, u_i \in [e]\} & [a] \triangleq \{a\} \end{array}$$

The key result about this fragment of the calculus of relations is the following characterisation: an equation is valid for relations if and only if it corresponds to an equality of languages.

▶ **Theorem 4.** *For all regular expressions e, f, we have*

$$\vDash e = f \qquad \text{iff} \qquad [e] = [f] \ .$$

We prove this theorem below. Its main consequence in practice is the decidability of the validity of equations: $[e]$ and $[f]$ are regular languages which we can easily represent using finite automata in order to compare them. This characterisation also gives us the precise complexity of the problem, as language equivalence of regular expression is PSPACE-complete [23].

Proof of Theorem 4. First we show the implication from left to right. Suppose $\vDash e = f$, we have to find a function σ from the alphabet Σ to a space of relations $\mathcal{P}(E{\times}E)$, such that $\hat{\sigma}(e) = \hat{\sigma}(f)$ entails $[e] = [f]$. Take $E = \Sigma^\star$, the set of words over Σ, and define σ as follows:

$$\begin{aligned}\sigma : \Sigma &\to \mathcal{P}(\Sigma^\star{\times}\Sigma^\star)\\ a &\mapsto \{\langle u, ua\rangle \mid u \in \Sigma^\star\}\end{aligned}$$

We will show that for all expression g, we have

$$\hat{\sigma}(g) = \{\langle u, uv\rangle \mid u \in \Sigma^\star, v \in [g]\} \ .$$

In particular, we will thus have $v \in [g]$ if and only if $\langle \epsilon, v\rangle \in \hat{\sigma}(g)$, so that $\hat{\sigma}(e) = \hat{\sigma}(f)$ entails $[e] = [f]$.

We proceed by induction on the expression g:

- $g = g' + g''$: we have

$$\begin{aligned}\hat{\sigma}(g) &= \hat{\sigma}(g') \cup \hat{\sigma}(g'')\\ &= \{\langle u, uv\rangle \mid u \in \Sigma^\star, v \in [g']\} \cup \{\langle u, uv\rangle \mid u \in \Sigma^\star, v \in [g'']\} && \text{(by induction)}\\ &= \{\langle u, uv\rangle \mid u \in \Sigma^\star, v \in [g'] \cup [g'']\}\\ &= \{\langle u, uv\rangle \mid u \in \Sigma^\star, v \in [g' + g'']\}\end{aligned}$$

- $g = g' \cdot g''$: we have

$$\begin{aligned}\hat{\sigma}(g) &= \hat{\sigma}(g') \cdot \hat{\sigma}(g'')\\ &= \{\langle u, uv\rangle \mid u \in \Sigma^\star, v \in [g']\} \cdot \{\langle u', u'w\rangle \mid u' \in \Sigma^\star, w \in [g'']\} && \text{(by induction)}\\ &= \{\langle u, uvw\rangle \mid u \in \Sigma^\star, v \in [g'], w \in [g'']\}\\ &= \{\langle u, uv\rangle \mid u \in \Sigma^\star, v \in [g' \cdot g'']\}\end{aligned}$$

- $g = g'^\star$: like in the previous point, we have

$$\begin{aligned}\hat{\sigma}(g) &= \hat{\sigma}(g')^\star\\ &= \{\langle u, uv\rangle \mid u \in \Sigma^\star, v \in [g']\}^\star && \text{(by induction)}\\ &= \{\langle u, uv\rangle \mid u \in \Sigma^\star, v \in [g'^\star]\}\end{aligned}$$

  (for the last step, we first show the following property, by induction on $k \in \mathbb{N}$: for all language $L \subseteq \Sigma^\star$, we have $\{\langle u, uv\rangle \mid u \in \Sigma^\star, v \in L\}^k = \{\langle u, uv\rangle \mid u \in \Sigma^\star, v \in L^k\}$).
- $g = 0$, $g = 1$, $g = a$: by unfolding definitions.

Now consider the converse implication. Fix a set E and a function $\sigma : \Sigma \to \mathcal{P}(E{\times}E)$; we have to show that $[e] = [f]$ entails $\hat{\sigma}(e) = \hat{\sigma}(f)$. This implication follows immediately from the following property, which we prove by induction on the expression g:

$$\hat{\sigma}(g) = \bigcup_{v \in [g]} \hat{\sigma}(v) \ .$$

(Note the slight abuse of notation in the term of the union: we apply the function $\hat{\sigma}$, expecting a regular expression, to a word v; we implicitely use the natural injection from words to expressions.)

- $g = g' + g''$: we have

$$\begin{aligned}
\hat\sigma(g) &= \hat\sigma(g') \cup \hat\sigma(g'') \\
&= \bigcup_{v\in[g']} \hat\sigma(v) \;\cup\; \bigcup_{v\in[g'']} \hat\sigma(v) && \text{(by induction)} \\
&= \bigcup_{v\in[g']\cup[g'']} \hat\sigma(v) \\
&= \bigcup_{v\in[g'+g'']} \hat\sigma(v)
\end{aligned}$$

- $g = g' \cdot g''$: we have

$$\begin{aligned}
\hat\sigma(g) &= \hat\sigma(g') \cdot \hat\sigma(g'') \\
&= \bigcup_{v\in[g']} \hat\sigma(v) \cdot \bigcup_{w\in[g'']} \hat\sigma(w) && \text{(by induction)} \\
&= \bigcup_{v\in[g'],\ w\in[g'']} \hat\sigma(v)\cdot\hat\sigma(w) && \text{(distributivity)} \\
&= \bigcup_{v\in[g'],\ w\in[g'']} \hat\sigma(v\cdot w) \\
&= \bigcup_{u\in[g'\cdot g'']} \hat\sigma(u)
\end{aligned}$$

- $g = g'^\star$: we have

$$\begin{aligned}
\hat\sigma(g) &= \hat\sigma(g')^\star \\
&= (\bigcup_{v\in[g']} \hat\sigma(v))^\star && \text{(by induction)} \\
&= \bigcup_{v_1\in[g'],\ldots,v_n\in[g']} \hat\sigma(v_1)\cdot\cdots\cdot\hat\sigma(v_n) \\
&= \bigcup_{v_1\in[g'],\ldots,v_n\in[g']} \hat\sigma(v_1\cdot\cdots\cdot v_n) \\
&= \bigcup_{u\in[g'^\star]} \hat\sigma(u)
\end{aligned}$$

- $g = 0$, $g = 1$, $g = a$: again, by unfolding definitions. ◀

Note that this proof leads to a similar characterisation for inequations: for all regular expressions e and f,

$$\vDash e \subseteq f \qquad \text{iff} \qquad [e] \subseteq [f] \ .$$

2.2 Axiomatisation

In 1956, Kleene asks for axiomatisations of the previous theory [7]: is it possible to find a small set of axioms (i.e., equations), from which follow all valid equations between regular expressions?

In the sixties, Salomaa gives two axiomatisations [22] which are not purely algebraic, and Redko proves that no finite equational axiomatisation can be complete [21]. Conway studies extensively this kind of questions in his monograph on regular algebra and finite

$$\left.\begin{aligned} e+(f+g) &= (e+f)+g \\ e+f &= f+e \\ e+0 &= e \\ e+e &= e \end{aligned}\right\} \langle +, 0\rangle \text{ is a commutative and idempotent monoid}$$

$$\left.\begin{aligned} e{\cdot}(f{\cdot}g) &= (e{\cdot}f){\cdot}g \\ e{\cdot}1 &= e \\ 1{\cdot}e &= e \end{aligned}\right\} \langle \cdot, 1\rangle \text{ is a monoid}$$

$$\left.\begin{aligned} e{\cdot}(f+g) &= e{\cdot}f + e{\cdot}g \\ (e+f){\cdot}g &= e{\cdot}g + f{\cdot}g \\ e{\cdot}0 &= 0 \\ 0{\cdot}e &= 0 \end{aligned}\right\} \text{distributivity between the two monoids}$$

$$\left.\begin{aligned} 1 + e{\cdot}e^\star &= e^\star \\ e{\cdot}f \leq f \quad &\Rightarrow \quad e^\star{\cdot}f \leq f \\ f{\cdot}e \leq f \quad &\Rightarrow \quad f{\cdot}e^\star \leq f \end{aligned}\right\} \text{laws about Kleene star}$$

Figure 1 The axioms of Kleene algebra.

automata [7], but we have to wait for the nineties for new results: Krob and Kozen independently show that one can axiomatise this theory in a finite way, but using axioms that are not just equations, but implications between equations. (We move from varieties to quasi-varieties.)

Krob's proof is long and difficult [15], but it provides a complete picture: first he gives a purely equational axiomatisation, infinite but with more structure than Salomaa's axioms. Then he shows that those infinitely many axioms can be derived from various finite axiomatisations involving implications between equations.

On the contrary, Kozen goes straight to the point and focuses on a specific finite axiomatisation (with implications). His proof is not simple either, but much shorter [13, 14].

▶ **Theorem 5** (Kozen'91, Krob'91). *For all regular expressions e, f, we have $[e] = [f]$ if and only if the equality $e = f$ is derivable from the axioms listed in Figure 1, where notation $e \leq f$ is a shorthand for $e + f = f$.*

These axioms can be decomposed into four groups: the first three correspond to the fact that we have an idempotent non-commutative semiring; the last group of axioms characterises the operation of reflexive-transitive closure, often called "Kleene star" in this context. This group is not entirely symmetric: the law $1 + e^\star{\cdot}e = e^\star$ is omitted as it can be derived from the other axioms. The last two axioms are implications; intuitively, they tell that if an expression f is invariant under composition with another expression e, then it is also invariant with $e^\star$. The expressive power of the axiomatisation mainly comes from those two implications: they make it possible to reason inductively on Kleene star, in a purely algebraic way.

One easily checks that each of these axioms is valid in the model of binary relations, but also when interpreting the expressions e, f, g as arbitrary languages. The converse implication from Theorem 5 follows from this remark: we prove only valid equations using those axioms.

The difficulty lies in the other implication: the completeness of these axioms, the fact that any valid equation might eventually be deduced from these axioms. We do not detail the proof here; a key step consists in showing that the set of matrices with coefficients in a Kleene algebra forms a new Kleene algebra (a Kleene algebra being a structure satisfying the axioms from Figure 1).

▶ **Exercise 6.** *Prove the following laws by using only Kleene algebras axioms:*

$$g + e\cdot f \leq f \Rightarrow e^\star\cdot g \leq f$$
$$g + f\cdot e \leq f \Rightarrow g\cdot e^\star \leq f$$
$$1 + e^\star\cdot e = e^\star$$
$$e\cdot f \leq g\cdot e \Rightarrow e\cdot f^\star \leq g^\star\cdot e$$
$$e\cdot f = g\cdot e \Rightarrow e\cdot f^\star = g^\star\cdot e$$
$$e\cdot(f\cdot e)^\star = (e\cdot f)^\star\cdot e$$
$$(e + f)^\star = e^\star\cdot(f^\star\cdot e)^\star$$

3 The strange fragment: allegories

Now consider a different fragment, where we only have composition, intersection, converse, and constants 1 and $\top$. For reasons to become clear in Section 4, we reuse letters u, v, w to denote the corresponding regular expressions, which we shall call *terms*:

$$u, v, w ::= u\cdot v \mid u\cap v \mid u^\circ \mid 1 \mid \top \mid a \qquad (a \in \Sigma)\ .$$

Modulo the presence of the constant $\top$, this fragment was studied by Andréka and Bredikhin [2], and by Freyd and Scedrov [11] under the name of (representable) *allegories*. We will see that one can decide the validity of inequations (and thus also equations) in this fragment, but that again, the corresponding theory is not finitely axiomatisable in a purely equational way.

3.1 Decidability

The key idea consists in characterising valid inequations by the existence of graph homomorphisms. More precisely, homomorphisms of directed and edge-labeled graphs with two distinguished vertices.

▶ **Definition 7** (Graph). A *graph* is a tuple $\langle V, E, \iota, o\rangle$, where V is a set of *vertices*, $E \subseteq V\times\Sigma\times V$ is a set of labelled edges, and $\iota, o \in V$ are two distinguished vertices, respectively called *input* and *output*.

We let letters G, H range over graphs and we define the following operations:

- $G\cdot H$ is the graph obtained by composing the two graphs in series, that is, by putting them one after the other and by merging the output of G with the input of H;
- $G\cap H$ is the graph obtained by composing the two graphs in parallel, that is, by putting them side by side and by merging their inputs and their outputs;
- G° is the graph obtained from G by exchanging input and output (without reversing edges);
- $\underline{1}$ is the graph without edges and with a single vertex $(\langle\{*\}, \emptyset, *, *\rangle)$;
- $\underline{\top}$ is the graph without edges and with two vertices, where input and output are distinct $(\langle\{\star, \bullet\}, \emptyset, \star, \bullet\rangle)$;

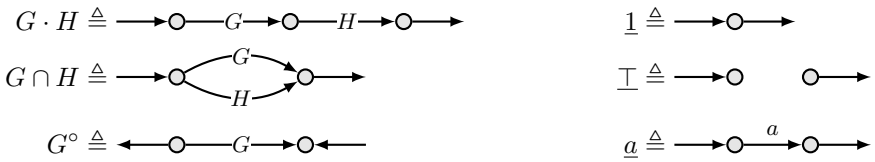


Figure 2 Operations on graphs.

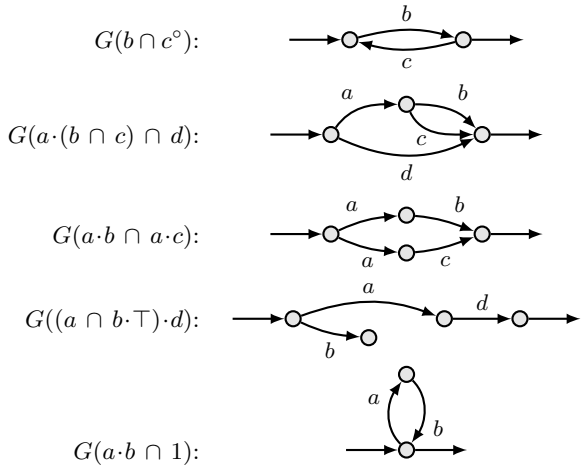


Figure 3 Graphs associated to some terms.

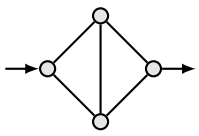
- for $a \in \Sigma$, $\underline{a}$ is the graph with two vertices and an edge labelled a from the input to the output $(\langle\{\star, \bullet\}, \{\langle\star, a, \bullet\rangle\}, \star, \bullet\rangle)$.

These operations are depicted on Figure 2; the input and the output of each graph is denoted using unlabelled arrows. These operations make it possible to associate a graph $G(u)$ to every term u, by structural induction:

$$G(u \cdot v) \triangleq G(u) \cdot G(v) \qquad G(1) \triangleq \underline{1}$$
$$G(u \cap v) \triangleq G(u) \cap G(v) \qquad G(\top) \triangleq \underline{\top}$$
$$G(u^\circ) \triangleq G(u)^\circ \qquad G(a) \triangleq \underline{a}$$

The graphs of a few terms are drawn on Figure 3. These are series-parallel graphs as long as we do not use converse and identity, that introduce loops in presence of intersection, nor the constant $\top$, that can disconnect some parts of the graphs.

Some graphs are not associated to any term. The canonical counter-example is the following one. (The labelling and the orientation of the five edges is irrelevant so that we omit this information.)



(8)

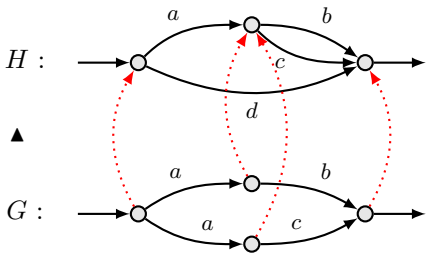


Figure 4 A graph homomorphism.

In fact, the graphs of terms are exactly the graphs of *treewidth* at most two; equivalently, they are the graphs excluding the complete graph with four vertices (K_4) as a minor[1] [9, 8].

One can compare graphs using homomorphisms:

▶ **Definition 8.** A *homomorphism* from the graph G to the graph H is a function from vertices of G to vertices of H that preserves labelled edges, input, and output. We write $H \blacktriangleleft G$ when there exists a homomorphism from G to H.

One easily checks that the relation $\blacktriangleleft$ is a preorder on graphs: it is reflexive and transitive.

As an example, the graph of $a{\cdot}(b \cap c) \cap d$ is smaller than that of $a{\cdot}b \cap a{\cdot}c$, thanks to the homomorphism depicted on Figure 4 using dotted arrows. Note that homomorphisms need not be injective or surjective, so that the preorder is completely unrelated to the sizes of the graphs: a graph may perfectly be smaller than another one, in the sense of the preorder, while having more vertices or edges (and vice-versa).

The nice property of the fragment considered here is the following characterisation: an inequation is valid for relations if and only if there exists a homorphism between the underlying graphs:

▶ **Theorem 9** ([2, Theorem 1], [11, page 208])**.** *For all terms u, v, we have*

$$\vDash u \subseteq v \qquad \textit{iff} \qquad G(u) \blacktriangleleft G(v) \ .$$

Graphs of terms being finite, one can look for a homomorphism between two such graphs in an exhaustive way, whence the decidability of the problem.

▶ **Exercise 10.** *Prove the laws* (1), (5), (6), *and* (7) *from Exercice 2, by using Theorem 9.*

We need a lemma in order to prove the theorem.

▶ **Lemma 11.** *Let u be a term, and let $G(u) = \langle V, E, \iota, o\rangle$ be its graph. Let S be a set and $\sigma : \Sigma \to \mathcal{P}(S{\times}S)$ an interpretation function. For all elements $i, j \in S$, we have $\langle i, j\rangle \in \hat{\sigma}(u)$ iff there exists a function $\phi : V \to S$ such that:*

$$\begin{cases} \phi(\iota) = i\,, \\ \phi(o) = j\,, \ \textit{et} \\ \langle p, a, q\rangle \in E \ \Rightarrow \ \langle \phi(p), \phi(q)\rangle \in \sigma(a) \ . \end{cases}$$

Proof. We proceed by induction on u:

[1] In both cases, after adding a edge between the input and the output.

- $u = v \cdot w$: write $G(v) = \langle V_v, E_v, \iota_v, o_v\rangle$ and $G(w) = \langle V_w, E_w, \iota_w, o_w\rangle$. We have $\langle i, j\rangle \in \hat{\sigma}(u) = \hat{\sigma}(v)\cdot\hat{\sigma}(w)$ iff there exists $k \in S$ such that $\langle i, k\rangle \in \hat{\sigma}(v)$ and $\langle k, j\rangle \in \hat{\sigma}(w)$. By induction, this last property is equivalent to the existence of two functions $\phi_u : V_u \to S$ et $\phi_v : V_v \to S$ such that $\phi_u(\iota_u) = i$, $\phi_u(o_u) = k$, $\langle p, a, q\rangle \in E_u$ entails $\langle \phi_u(p), \phi_u(q)\rangle \in \sigma(a)$, $\phi_v(\iota_v) = k$, $\phi_v(o_v) = j$, and $\langle p, a, q\rangle \in E_v$ entails $\langle \phi_v(p), \phi_v(q)\rangle \in \sigma(a)$. By gluing back those two functions, we easily show the equivalence with the existence of a function from the graph $G(u) = G(v)\cdot G(w)$ satisfying the property from the statement.
- $u = v \cap w$: with the notations from the previous point, we have $\langle i, j\rangle \in \hat{\sigma}(u) = \hat{\sigma}(v)\cap\hat{\sigma}(w)$ iff $\langle i, j\rangle \in \hat{\sigma}(v)$ and $\langle i, j\rangle \in \hat{\sigma}(w)$. By induction, this conjunction is equivalent to the existence of two functions $\phi_u : V_u \to S$ and $\phi_v : V_v \to S$ such that $\phi_x(\iota_x) = i$, $\phi_x(o_x) = j$, and $\langle p, a, q\rangle \in E_x$ entails $\langle \phi_x(p), \phi_x(q)\rangle \in \sigma(a)$, for $x \in \{u, v\}$. As previously one easily shows the equivalence with the existence of a function from the graph $G(u) = G(v)\cap G(w)$ satisfying the property from the statement.
- $u = 1$: by definition, we have $\langle i, j\rangle \in \hat{\sigma}(u) = 1$ iff $i = j$, and the existence of a function ϕ satisfying the properties of the statement for the graph $\underline{1}$ is also equivalent to $i = j$.
- $u = \top$: by definition, $\langle i, j\rangle \in \hat{\sigma}(u) = \top$ is always true; and the existence of a function ϕ satisfying the properties of the statement for the graph $\underline{\top}$ is always guaranteed.
- $u = a$: $\hat{\sigma}(u) = \sigma(a)$ the existence of a function ϕ satisfying the properties of the statement for the graph $\underline{a}$ is equivalent to the membership of $\langle i, j\rangle$ to $\sigma(a)$. ◀

Proof of Theorem 9. Write $G(u) = \langle V, E, \iota, o\rangle$ et $G(v) = \langle V', E', \iota', o'\rangle$.

Start by the right-to-left implication: assume $G(u) \blacktriangleleft G(v)$, i.e., a homomorphism γ from $G(v)$ to $G(u)$, and let us show $\vDash u \subseteq v$. Let S be a set and $\sigma : \Sigma \to \mathcal{P}(S\times S)$ an interpretation function; for all $\langle i, j\rangle \in \hat{\sigma}(u)$ (†), we have to show $\langle i, j\rangle \in \hat{\sigma}(v)$ (‡). Let $\phi : V \to S$ be the function given by Lemma 11 and assumption (†). By the same lemma, to prove (‡) it suffices to find a function $\psi : V' \to S$ satisfying $\psi(\iota') = i$, $\psi(o') = j$, and $\langle p', a, q'\rangle \in E'$ entails $\langle \psi(p'), \psi(q')\rangle \in \sigma(a)$. The composed function $\phi \circ \gamma$ is suitable.

Now let us show the direct implication. Suppose that $\vDash u \subseteq v$, we have to find a homomorphism from $G(v)$ to $G(u)$. Let σ be the following interpretation function:

$$\begin{aligned}\sigma : \Sigma &\to \mathcal{P}(V\times V)\\ a &\mapsto \{\langle p, q\rangle \mid \langle p, a, q\rangle \in E\}\end{aligned}$$

By Lemma 11, using the identity function, we have $\langle \iota, o\rangle \in \hat{\sigma}(u)$. By assumption, we deduce $\langle \iota, o\rangle \in \hat{\sigma}(v)$, whence, by using Lemma 11 again, the existence of a function $\phi : V' \to V$ satisfying some properties. These properties precisely correspond to the fact that ϕ is a homomorphism from $G(v)$ to $G(u)$. ◀

3.2 Axiomatisation

Freyd and Scedrov define *allegories* [11] as structures satisfying the axioms from Figure 5[2]. First note that composition does not distribute over intersections: composition is monotone in its two arguments, which entails the following inequations but not their converses:

$$e \cdot (f \cap g) \subseteq e\cdot f \cap e\cdot g$$
$$(f \cap g) \cdot e \subseteq f\cdot e \cap g\cdot e$$

[2] Up-to some details: they do not consider the constant $\top$, and they work in a categorical setting, where the various operations are typed.

$$\left.\begin{array}{c} e \cap (f \cap g) = (e \cap f) \cap g \\ e \cap f = f \cap e \\ e \cap \top = e \\ e \cap e = e \end{array}\right\} \langle \cap, \top \rangle \text{ is a commutative and idempotent monoid}$$

$$\left.\begin{array}{c} e \cdot (f \cdot g) = (e \cdot f) \cdot g \\ e \cdot 1 = e \\ 1 \cdot e = e \end{array}\right\} \langle \cdot, 1 \rangle \text{ is a monoid}$$

$$\left.\begin{array}{c} e \cdot (f \cap g) \subseteq e \cdot f \\ (f \cap g) \cdot e \subseteq f \cdot e \end{array}\right\} \text{composition is monotone}$$

$$\left.\begin{array}{c} e^{\circ\circ} = e \\ (e \cap f)^\circ \subseteq e^\circ \\ (e \cdot f)^\circ \subseteq f^\circ \cdot e^\circ \end{array}\right\} \begin{array}{l} \text{converse is a monotone} \\ \text{involution reversing composition} \end{array}$$

$$\left.\begin{array}{c} e \cdot f \cap g \subseteq (e \cap g \cdot f^\circ) \cdot f \end{array}\right\} \text{modularity law}$$

Figure 5 Axioms of allegories.

One can also deduce from the axioms that converse reverses composition, distributes over intersections, and preserves constants 1 et $\top$:

$$(e \cap f)^\circ = e^\circ \cap f^\circ \qquad \top^\circ = \top$$
$$(e \cdot f)^\circ = f^\circ \cdot e^\circ \qquad 1^\circ = 1$$

The last axiom in Figure 5 is uncommon. It is called *modularity law*, it is equivalent in presence of the other axioms to its symmetrical counterpart:

$$e \cdot f \cap g \subseteq e \cdot (f \cap e^\circ \cdot g)$$

It also admits as a consequence the following inequation, known as *Dedekind's inequality*:

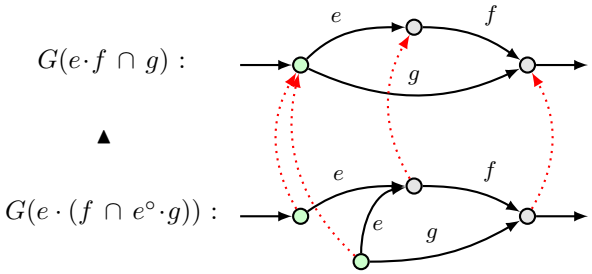
$$e \cdot f \cap g \subseteq (e \cap g \cdot f^\circ) \cdot (f \cap e^\circ \cdot g)$$

▶ **Exercise 12.** *Prove the six laws above from the axioms of Figure 5.*

Unfortunately, this finite and purely equational axiomatisation is not complete for relations: some valid equations are not consequences of the axioms. Freyd and Scedrov actually proved that there exists no finite equational axiomatisation. We give some intuitions about this result in the remainder of this section. Let us first check that the axiomatisation is sound:

▶ **Exercise 13.** *Prove that each axiom is valid by using Theorem 9: draw each graph and make explicit the homomorphisms corresponding to each inequation.*

When doing the above exercise, one can see that the only non-injective homomorphism is the one corresponding to the modularity law, and that this homomorphism equates exactly two vertices:



We actually have the following result:

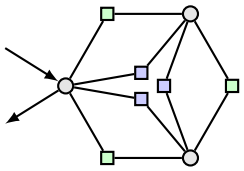
▶ **Claim 14.** *Let u and v be two terms. If there exists a homomorphism from $G(v)$ to $G(u)$ equating at most two vertices, then the inequality $u \subseteq v$ is a consequence of the axioms from Figure 5.*

Proof. Left to the reader by Freyd and Scedrov [11]. ◀

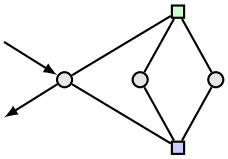
The converse does not hold: many inequations provable from the axioms correspond to homomorphisms equating arbitrarily many vertices (for instance, Dedekind's inequality, where two pairs of vertices are equated, or the inequation (1) from Exercice 2, where the five vertices of the right-hand side are equated).

Consider nevertheless an arbitrary homomorphism from the graph of a term v to that of a term u. This homomorphism can be decomposed in several ways into a sequence of homomorphisms each equating at most two vertices. One could thus believe that it suffices to use the claim 14 to obtain a sequence of provable inequations, leading to a proof of $u \subseteq v$ from the axioms and transitivity.

The problem is that the intermediate graphs appearing in these sequences of homomorphisms need not be graphs of terms (recall the graph (8)). Here is a counter-example; again, we do not label the edges nor we give their orientation as this information is irrelevant. Consider the following graph:



This graph corresponds to a term of the shape $1 \cap \Pi_{i=1,2,3}(a_i \cdot b_i \cap c_i \cdot d_i)$. If we equate the three inner, square, blue vertices, as well as the three outer, square, green vertices, we obtain the following graph:



This graph is associated to a term, of the shape $1 \cap i \cdot (e \cdot f \cap g \cdot h) \cdot j$, so that the homomorphism implicitely considered corresponds to a valid inequation between two terms.

This homomorphism equates in one step two groups of three vertices. Now let us try to decompose it into a sequence of four morphisms equating each exactly two vertices. The first homomorphism must equate two blue vertices, or two green vertices. In both cases, we obtain a graph which is not the graph of any term.

By formalising this idea more precisely, one obtains a valid inequation which cannot be proved from the axioms, whence the incompletness of the axiomatisation. One can actually generalise the counter-example and show that every complete equational axiomatisation must contain axioms corresponding to homomorphisms equating arbitrarily many vertices, whence the impossibility for this axiomatisation to be finite [11, page 210].

Hodkinson and Mikulás further showed that there cannot be a finite first-order axiomatisation [12], and in particular a quasi-equational one like, e.g., for Kleene algebra. In contrast, we proved recently with Cosme-Llópez that the more restrictive theory of *isomorphism* (on graphs of terms) can be finitely axiomatised in a purely equational way [8].

4 Putting all together

Let us come back to the initial problem, that of the positive calculus of relations. We have seen that two fragments are decidable: the fragment corresponding to regular expressions $(+,\cdot,\cdot^\star,0,1)$, and that corresponding to allegories $(\cap,\cdot,\cdot^\circ,\top,1)$. What happens when we take all operations?

First note that the function $[\cdot]$ associating a (regular) language to every regular expression can be extended to the operations of allegories:

$$\begin{aligned}[e\cap f] &\triangleq [e]\cap[f]\\ [e^\circ] &\triangleq \{a_n\dots a_1 \mid a_1\dots a_n\in[e]\}\\ [\top] &\triangleq \Sigma^\star\end{aligned}$$

However, the characterisation obtained in Theorem 4 no longer works with these operations. Indeed, we have for instance

$$\begin{aligned}[a\cap b]=\{a\}\cap\{b\}=\emptyset=[0] \quad&\text{but}\quad \not\vdash a\cap b=0\\ [a^\circ]=\{a\}=[a] \quad&\text{but}\quad \not\vdash a^\circ=a\\ [a]=\{a\}\not\subseteq\{aaa\}=[a\cdot a^\circ\cdot a] \quad&\text{but}\quad \vdash a\subseteq a\cdot a^\circ\cdot a\\ [\top\cdot a\cdot\top\cdot b\cdot\top]\neq[\top\cdot b\cdot\top\cdot a\cdot\top] \quad&\text{but}\quad \vdash \top\cdot a\cdot\top\cdot b\cdot\top=\top\cdot b\cdot\top\cdot a\cdot\top\end{aligned}$$

To obtain a characterisation, we actually have to replace words (elements of $\Sigma^\star$) by graphs, and thus consider languages of graphs.

▶ **Definition 15.** The *language of graphs* of an expression e, written $\mathcal{G}(e)$, is defined as follows, by induction on e:

$$\begin{aligned}\mathcal{G}(e+f)&\triangleq\mathcal{G}(e)\cup\mathcal{G}(f) & \mathcal{G}(0)&\triangleq\emptyset\\ \mathcal{G}(e\cap f)&\triangleq\{G\cap H\mid G\in\mathcal{G}(e),\ H\in\mathcal{G}(f)\} & \mathcal{G}(\top)&\triangleq\{\underline{\top}\}\\ \mathcal{G}(e\cdot f)&\triangleq\{G\cdot H\mid G\in\mathcal{G}(e),\ H\in\mathcal{G}(f)\} & \mathcal{G}(1)&\triangleq\{\underline{1}\}\\ \mathcal{G}(e^\star)&\triangleq\{G_1\cdot\dots\cdot G_n\mid n\in\mathbb{N},\ \forall i\leq n,,G_i\in\mathcal{G}(e)\} & \mathcal{G}(a)&\triangleq\{\underline{a}\}\\ \mathcal{G}(e^\circ)&\triangleq\{G^\circ\mid G\in\mathcal{G}(e)\}\end{aligned}$$

This definition properly generalises the usual notion of language: when the considered expression contains no intersection, no converse, and no constant $\top$, then the associated graphs are isomorphic to words: these are simple threads labelled by letters in Σ.

To generalise also allegories, we have to make use of graph homomorphisms. Given a set L of graphs, we write ${}^\blacktriangleleft L$ for is *downward closure* w.r.t. the preorder $(\blacktriangleleft)$:

$${}^\blacktriangleleft L\triangleq\{G\mid\exists H, G\blacktriangleleft H,\ H\in L\}\ .$$

We finally obtain the following characterisation:

▶ **Theorem 16.** *For all expressions e and f, we have*

$$\vDash e \subseteq f \qquad \textit{iff} \qquad \mathcal{G}(e) \subseteq {}^{\blacktriangleleft}\mathcal{G}(f) \ .$$

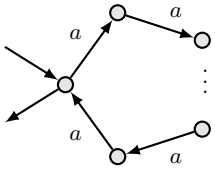
Proof. Similar to the proof of Theorem 4, using Theorem 9 (cf. [5, Theorem 6], adding the constant $\top$ is not problematic). ◀

This characterisation generalises both Theorem 4 and Theorem 9. If e and f are regular expressions, then all graphs in $\mathcal{G}(e)$ and $\mathcal{G}(f)$ are threads, and the unique possible homomorphism between two such graphs is the identity; whence $\mathcal{G}(e) \subseteq {}^{\blacktriangleleft}\mathcal{G}(f)$ iff $\mathcal{G}(e) \subseteq \mathcal{G}(f)$. If instead e and f are terms u and v, then $\mathcal{G}(e) = \{G(u)\}$ and $\mathcal{G}(f) = \{G(v)\}$, so that $\mathcal{G}(e) \subseteq {}^{\blacktriangleleft}\mathcal{G}(f)$ is equivalent to $G(u) \blacktriangleleft G(v)$.

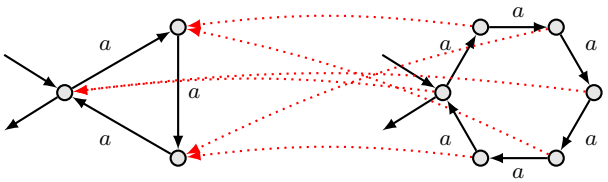
Note also that for all graph languages L, K, we have $L \subseteq {}^{\blacktriangleleft}K$ iff ${}^{\blacktriangleleft}L \subseteq {}^{\blacktriangleleft}K$. Valid equations are thus characterised as follows:

$$\vDash e = f \qquad \text{iff} \qquad {}^{\blacktriangleleft}\mathcal{G}(e) = {}^{\blacktriangleleft}\mathcal{G}(f) \ .$$

To illustrate this theorem, consider expressions $e \triangleq a^+ \cap 1$ and $f \triangleq (a{\cdot}a)^+ \cap 1$, where g^+ is a shorthand for $g{\cdot}g^\star$. The set of graphs $\mathcal{G}(e)$ is the set of non-trivial cycles labelled with a:



On the other side, $\mathcal{G}(f)$ is the set of non-trivial cycles of even length. Thus we immediately get $\mathcal{G}(f) \subseteq \mathcal{G}(e) \subseteq {}^{\blacktriangleleft}\mathcal{G}(e)$, whence $\vDash f \subseteq e$. The converse inequation is also valid: to each cycle from $\mathcal{G}(e)$, possibly of odd length, one can associate the cycle of double length, in $\mathcal{G}(f)$; indeed, there is a homomorphism from this cycle of double length into the shorter one:



▶ **Exercise 17.** *Use the same technique to prove the following laws:*

$$(a \cap b{\cdot}b)^\star \subseteq a^\star \cap b^\star$$
$$((a \cap b){\cdot}(1 \cap b){\cdot}(a \cap b))^\star \subseteq (a \cap b{\cdot}b)^\star$$
$$(a \cap b{\cdot}\top)^\star{\cdot}(1 \cap b{\cdot}\top) = (1 \cap \top{\cdot}b^\circ){\cdot}(a \cap \top{\cdot}b^\circ)^\star$$

Together with Paul Brunet [5], we proposed an automata model allowing us to recognise languages of graphs associated to expressions. This automata model takes inspiration from Petri nets [19, 17], which make it possible to explore richer structures than plain words. To each expression e, we associate what we call a *Petri automaton*, whose language is precisely ${}^{\blacktriangleleft}\mathcal{G}(e)$. Thanks to Theorem 16, the problem of validity of equations or inequations thus reduces to the problem of comparing Petri automata.

We solved this algorithmic problem only for a fragment of the calculus: we have to forbid converse and constants 1 and $\top$, and replace reflexive-transitive closure $\cdot^\star$ by transitive closure $\cdot^+$ (because reflexive-transitive closure implicitly contains the identity: we have $1 = 0^\star$). The corresponding equational theory was recently studied by Andréka, Mikulás, and Németi [1]: it coincide with that of languages over this signature. Under this restriction, the considered graphs are always acyclic, so that the automata become simpler to compare: we have shown that the problem of comparing these automata is EXPSPACE-complete [5].

Subsequently, Nakamura managed to prove that the problem remains in EXPSPACE in presence of converse and identity (but without $\top$, although his technique certainly applies) [18]. His solution consists in defining a notion of partial derivatives for graphs, similar to Antimirov' partial derivatives for regular expressions [3], and exploiting the fact that graph generated from a given expression have a bounded pathwidth [9].

5 Open questions

Is it possible to axiomatise the positive calculus of relations with transitive closure? For instance, do Kleene algebra axioms suffice when added to a complete axiomatisation of representable allegories? What about the fragment without converse, identity, and $\top$, studied by Andréka, Mikulás, and Németi [1]?

Note that intersection is the difficult operation: without intersection (and associated constant $\top$), we obtain *Kleene algebras with converse*, for which Bernátsky, Bloom, Ésik and Stefanescu have obtained decidability [4][3] and complete axiomatisability relatively to Kleene algebras: the following five axioms suffice when added to any complete axiomatisation of Kleene algebras (e.g., those from Figure 1) [10].

$$(e \cdot f)^\circ = f^\circ \cdot e^\circ \qquad e^{\circ\star} = e^{\star\circ} \qquad e \subseteq e \cdot e^\circ \cdot e$$
$$(e + f)^\circ = e^\circ + e^\circ \qquad e^{\circ\circ} = e$$

References

1 Hajnal Andréka, Szabolcs Mikulás, and István Németi. The equational theory of kleene lattices. *Theor. Comput. Sci.*, 412(52):7099–7108, 2011. doi:10.1016/j.tcs.2011.09.024.

2 H. Andréka and D.A. Bredikhin. The equational theory of union-free algebras of relations. *Algebra Universalis*, 33(4):516–532, 1995. doi:10.1007/BF01225472.

3 Valentin M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theor. Comput. Sci.*, 155(2):291–319, 1996. doi:10.1016/0304-3975(95)00182-4.

4 Stephen L. Bloom, Zoltán Ésik, and Gheorghe Stefanescu. Notes on equational theories of relations. *algebra universalis*, 33(1):98–126, Mar 1995. doi:10.1007/BF01190768.

5 Paul Brunet and Damien Pous. Petri automata for kleene allegories. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 68–79. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.17.

6 Paul Brunet and Damien Pous. Algorithms for kleene algebra with converse. *J. Log. Algebr. Meth. Program.*, 85(4):574–594, 2016. doi:10.1016/j.jlamp.2015.07.005.

7 J. H. Conway. *Regular algebra and finite machines*. Chapman and Hall, 1971.

[3] Which we refined to PSPACE-completeness with Brunet [6].

8 Enric Cosme-Llópez and Damien Pous. K4-free graphs as a free algebra. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, volume 83 of *LIPIcs*, pages 76:1–76:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.MFCS.2017.76.

9 R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer Verlag, 2005.

10 Zoltán Ésik and L. Bernátsky. Equational properties of kleene algebras of relations with conversion. *Theor. Comput. Sci.*, 137(2):237–251, 1995. doi:10.1016/0304-3975(94)00041-G.

11 P. Freyd and A. Scedrov. *Categories, Allegories*. North Holland, 1990.

12 Ian Hodkinson and Szabolcs Mikulás. Axiomatizability of reducts of algebras of relations. *Algebra Universalis*, 43(2):127–156, 2000. doi:10.1007/s000120050150.

13 Dexter Kozen. A completeness theorem for kleene algebras and the algebra of regular events. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 214–225. IEEE Computer Society, 1991. doi:10.1109/LICS.1991.151646.

14 Dexter Kozen. A completeness theorem for kleene algebras and the algebra of regular events. *Inf. Comput.*, 110(2):366–390, 1994. doi:10.1006/inco.1994.1037.

15 Daniel Krob. Complete systems of b-rational identities. *Theor. Comput. Sci.*, 89(2):207–343, 1991. doi:10.1016/0304-3975(91)90395-I.

16 Donald Monk. On representable relation algebras. *Michigan Math. J.*, 11(3):207–210, 09 1964. doi:10.1307/mmj/1028999131.

17 T. Murata. Petri nets: Properties, analysis and applications. *Proc. of the IEEE*, 77(4):541–580, Apr 1989. doi:10.1109/5.24143.

18 Yoshiki Nakamura. Partial derivatives on graphs for kleene allegories. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005132.

19 C. A. Petri. Fundamentals of a theory of asynchronous information flow. In *IFIP Congress*, pages 386–390, 1962.

20 Damien Pous. *Informatique Mathématique: une photographie en 2016*, chapter Algèbres de relations. E. Baudrier and L. Mazo (Eds), 2016. Course notes for the EJCIM research school.

21 V. Redko. On defining relations for the algebra of regular events. *Ukr. Mat. Z.*, 16:120–, 1964.

22 Arto Salomaa. Two complete axiom systems for the algebra of regular events. *J. ACM*, 13(1):158–169, 1966. doi:10.1145/321312.321326.

23 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 1–9. ACM, 1973. doi:10.1145/800125.804029.

24 A. Tarski and S. Givant. *A Formalization of Set Theory without Variables*, volume 41 of *Colloquium Publications*. American Mathematical Society, Providence, Rhode Island, 1987.

25 Alfred Tarski. On the calculus of relations. *J. Symbolic logic*, 6:73–89, 1941.

The Open Shop Scheduling Problem

Gerhard J. Woeginger
Department of Computer Science, RWTH Aachen, D-52074 Aachen, Germany
woeginger@algo.rwth-aachen.de

Abstract

We discuss the computational complexity, the approximability, the algorithmics and the combinatorics of the open shop scheduling problem. We summarize the most important results from the literature and explain their main ideas, we sketch the most beautiful proofs, and we also list a number of open problems.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms, Mathematics of computing → Discrete optimization

Keywords and phrases Algorithms, Complexity, Scheduling, Approximation

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.4

Category Invited Talk

1 Blacksmiths and horseshoes

"There are four blacksmiths working together. One of them has specialized in putting horseshoes on the left front leg of a horse, while the other three have specialized respectively in putting horseshoes on the left hind leg, the right front leg, and the right hind leg. If the work on one horseshoe takes five minutes, what is the minimum amount of time needed to put twenty-eight horseshoes on seven horses? (Note that a horse cannot stand on two legs.)"

As each blacksmith has to handle 7 horseshoes, he needs at least 35 minutes of working time. The following picture with horses A, B, C, D, E, F, G and blacksmiths $1, 2, 3, 4$ shows a schedule that meets this lower bound of 35 minutes. Note that each horse receives its four horseshoes during four different time slots (so that it never has to stand on two legs), and note that during each five minute time slot each blacksmith works for exactly five non-interrupted minutes on a single horse.

	minute 00–05	minute 05–10	minute 10–15	minute 15–20	minute 20–25	minute 25-30	minute 30-35
Blacksmith 1	**A**	**B**	**C**	**D**	**E**	**F**	**G**
Blacksmith 2	**B**	**C**	**D**	**G**	**F**	**E**	**A**
Blacksmith 3	**C**	**D**	**G**	**E**	**A**	**B**	**F**
Blacksmith 4	**D**	**A**	**F**	**B**	**C**	**G**	**E**

2 Problem statement and some definitions

An instance of the *open shop* scheduling problem consists of m machines $M_1, \ldots, M_m$ and n jobs $J_1, \ldots, J_n$. (Throughout, machines will be indexed by i and jobs will be indexed by j.) Each job J_j consists of m independent operations $O_{i,j}$ with $i = 1, \ldots, m$. The operation $O_{i,j}$ of job J_j has to be processed on machine M_i, which takes $p_{i,j}$ uninterrupted time units. For

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 4; pp. 4:1–4:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

every job, the order in which its operations have to be processed is *not fixed* in advance but may be chosen arbitrarily by the scheduler; we stress that different jobs may receive different processing orders.

A *schedule* assigns every operation $O_{i,j}$ to a time interval of length $p_{i,j}$, so that no job is simultaneously processed on two different machines and so that no machine simultaneously processes two different jobs. The *makespan* $C_{\max}$ of a schedule is the largest job completion time. The optimal makespan is usually denoted by $C^*_{\max}$. In the standard scheduling classification scheme of Lawler, Lenstra, Rinnooy Kan & Shmoys [12], this optimization problem is denoted by $O||C_{\max}$ (if the number m of machines is given as part of the input) and by $Om||C_{\max}$ (if the number m of machines is a fixed constant number).

In the *"Blacksmiths and horseshoes"* puzzle, the four blacksmiths are four machines $M_1, M_2, M_,M_4$. Each horse forms a job, and its four legs are the four operations of that job. All operations $O_{i,j}$ have length $p_{i,j} = 5$.

Here are some more notations. The length of the longest operation in an instance is denoted $o_{\max} = \max_{i,j} p_{i,j}$. The overall processing time of job J_j will be denoted $p_j = \sum_{i=1}^{m} p_{i,j}$. The overall processing time assigned to machine M_i is called the *load* $L_i = \sum_{j=1}^{n} p_{i,j}$ of the machine. The maximum job processing time is denoted $p_{\max} = \max_j p_j$ and the maximum machine load is denoted $L_{\max} = \max_i L_j$. As no job can be simultaneously processed on two different machines the makespan of any schedule satisfies $C_{\max} \geq p_{\max}$, and as no machine can simultaneously processes two different jobs the makespan satisfies $C_{\max} \geq L_{\max}$. This yields the following lower bound, which will be important throughout the paper:

$$C^*_{\max} \;\geq\; \beta^* \;:=\; \max\{L_{\max},\, p_{\max}\} \tag{1}$$

We mention in passing that there are two other important shop scheduling problems that are closely related to the open shop problem: In a *flow shop*, every job must pass the machines in the same ordering $M_1, \ldots, M_m$. In a *job shop*, the ordering of the operations is fixed a priori for every job, and different jobs may have different orderings of operations. These variants will not be further discussed in the rest of this paper.

3 Computational complexity

Gonzalez & Sahni [10] prove that the open shop on $m = 2$ machines allows a very simple polynomial time solution: There always exists a schedule whose makespan equals the lower bound β^* in (1), and this schedule can be found in linear time.

▶ **Theorem 1** (Gonzalez & Sahni [10]). *Problem $O2||C_{\max}$ is solvable in polynomial time.*

The algorithm in Theorem 1 is not hard to find (there are many possible approaches), and we leave it as a puzzle for the reader. A more general problem variant considers the completion time C_1 of the last operation on machine M_1 and the completion time C_2 of the last operation on M_2, and asks for a schedule that minimizes some objective function $f(C_1, C_2)$ of the two machine completion times. Based on extensive case distinctions, Shaklevich & Strusevich [21] develop linear time algorithms for this variant, if the function $f(\cdot, \cdot)$ is non-decreasing in both arguments. Van den Akker, Hoogeveen & Woeginger [23] provide a simpler proof of the same result. Sahni & Cho [14] prove strong NP-hardness of the no-wait problem $O2|\text{no-wait}|C_{\max}$ in which the processing of the second operation of each job must start immediately after the completion of its first operation.

Now let us turn to the cases of $Om||C_{\max}$ with $m \geq 3$ machines. As usual, the complexity jumps from easy to hard when we move from parameter 2 to parameter 3:

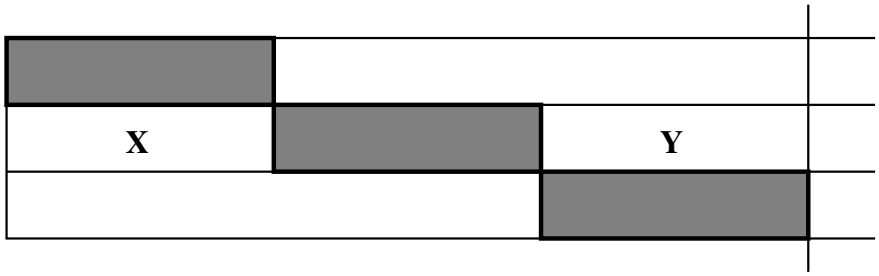


Figure 1 Illustration of the NP-hardness argument in Theorem 2.

▶ **Theorem 2** (Gonzalez & Sahni [10]). *For every fixed $m \geq 3$, problem $Om||C_{\max}$ is NP-hard.*

Proof. We only show hardness for $m = 3$. The proof is a polynomial time reduction from the NP-hard PARTITION problem [9]: *"Given k positive integers $q_1, \ldots, q_k$ with $\sum_{i=1}^{k} q_i = 2Q$, does there exist an index set $I \subseteq \{1, \ldots, k\}$ with $\sum_{i \in I} q_i = Q$?"*

For $j = 1, \ldots, k$ we create a job J_j with $p_{1,j} = p_{2,j} = p_{3,j} = q_j$. Furthermore, there is a job J_{k+1} with $p_{1,k+1} = p_{2,k+1} = p_{3,k+1} = Q$. We claim that the PARTITION instance has answer YES, if and only if the constructed instance of $O3||C_{\max}$ has a schedule with makespan at most $3Q$. The (only if part) is straightforward. For the (if part), consider the three operations of job J_{k+1} in a schedule with makespan $3Q$. By symmetry, we may assume that J_{k+1} is first processed on machine M_1, then on M_2, and finally on M_3. Then the second operation generates two time intervals X and Y of length Q on machine M_2; see Figure 1 for an illustration. The operations $O_{2,j}$ of the other jobs must be packed into intervals X and Y, and thereby yield a solution for the PARTITION instance. ◀

As the PARTITION problem is NP-hard in the weak sense, the argument in Theorem 2 only yields NP-hardness in the weak sense for $Om||C_{\max}$. The precise complexity (strong NP-hardness versus pseudo-polynomial time solvability) of problem $Om||C_{\max}$ is unknown. This complexity question has been open since the 1970s, and it forms the biggest and most important gap in our understanding of open shop scheduling.

▶ **Open problem 3.** *Prove that for every fixed number $m \geq 3$ of machines, problem $Om||C_{\max}$ is solvable in pseudo-polynomial time.*

Finally, let us discuss the complexity of problem $O||C_{\max}$ where the number of machines is specified as part of the input. An unpublished result of Lenstra [13] from the 1970s establishes strong NP-hardness of $O||C_{\max}$. Strong NP-hardness of $O||C_{\max}$ can also easily be deduced from a published result by Williamson & al. [24], who prove that $O||C_{\max}$ is NP-hard, even if all operations are of length $0, 1, 2$ and if the question is to decide whether there is a schedule with makespan 4.

4 A theorem on vector rearrangements

This section introduces an auxiliary problem and an auxiliary result that will be pivotal for the next section. Let $B \subset \mathbb{R}^d$ be the unit ball of a norm $\|\cdot\|$ on $\mathbb{R}^d$, that is, a d-dimensional closed convex body that is centrally symmetric about the origin. Suppose we are given n vectors $v_1, \ldots, v_n \in \mathbb{R}^d$ that satisfy

$$\sum_{i=1}^{n} v_i = 0 \qquad \text{and} \qquad \|v_i\| \leq 1 \text{ for } 1 \leq i \leq n. \tag{2}$$

The goal is to find a permutation $v_{\pi(1)}, \ldots, v_{\pi(n)}$ of these vectors, so that for every k with $1 \leq k \leq n$ the norm $\|v_{\pi(1)} + v_{\pi(2)} + \cdots + v_{\pi(k)}\|$ of the partial sum is small. Steinitz [22] proved in 1913 that the norms of these partial sums can be bounded by a constant that only depends on the unit ball B (and Steinitz showed that the concrete constant $2d$ always works). The smallest such constant is called the *Steinitz constant* $c(B)$ of the norm.

▶ **Theorem 4** (Grinberg & Sevastianov [11]). *For any norm with unit ball $B \subset \mathbb{R}^d$, the Steinitz constant satisfies $c(B) \leq d$.*

The proof of Theorem 4 in [11] is an optimized and streamlined version of an earlier proof by Sevastianov [15]. It is extremely elegant, and we will sketch it now. Hence, let us consider vectors $v_1, \ldots, v_n \in \mathbb{R}^d$ that satisfy (2). In a first step, we prove that there exists a system of subsets $V_d, V_{d+1}, \ldots, V_n$ of $\{v_1, \ldots, v_n\}$ that satisfies the following properties.

- $V_d \subseteq V_{d+1} \subseteq \cdots \subseteq V_n = \{v_1, \ldots, v_n\}$
- $|V_k| = k$ for $1 \leq k \leq n$
- There exist real numbers $\lambda_k(v) \in [0, 1]$ for $d \leq k \leq n$ and $v \in V_k$ with
  (A) $\sum_{v \in V_k} \lambda_k(v) = k - d$ for $d \leq k \leq n$, and
  (B) $\sum_{v \in V_k} \lambda_k(v) \cdot v = 0$ for $d \leq k \leq n$.

In other words, the coefficients $\lambda_k(v)$ constitute a linear dependency on V_k where all coefficients add up to $k - d$. The subsets V_k and the real numbers $\lambda_k(v)$ are constructed by a backward induction. For $k = n$, we have $V_n = \{v_1, \ldots, v_n\}$ and we define $\lambda_n(v) \equiv (n - d)/n$ for all v. These values satisfy condition (A) by definition, while condition (B) follows from (2).

Now assume that the sets $V_{k+1}, \ldots, V_n$ have already been defined together with the corresponding coefficients $\lambda_{k+1}(v), \ldots, \lambda_n(v)$. We consider the following system of linear constraints on $k + 1$ real variables $x(v)$ for $v \in V_{k+1}$.

$$\sum_{v \in V_{k+1}} x(v) \;=\; k - d \tag{3}$$

$$\sum_{v \in V_{k+1}} x(v) \cdot v \;=\; 0 \tag{4}$$

$$0 \leq x(v) \leq 1 \qquad \text{for } v \in V_{k+1} \tag{5}$$

Note that the system (3)–(5) is solvable, as can be seen for instance by setting

$$x(v) \;=\; \frac{k - d}{k + 1 - d}\, \lambda_{k+1}(v) \qquad \text{for } v \in V_{k+1}.$$

Hence the underlying $(k + 1)$-dimensional polytope is non-empty. Any extreme point x^* of this polytope must satisfy $k + 1$ of the linear constraints with equality. As constraint (3) yields one such equality and as constraint (4) yields d such equalities (one per dimension), in an extreme point at least $k + 1 - (d + 1) = k - d$ of the inequalities in (5) must be tight. Because of (3), this implies that in an extreme point x^* at least one of the variables $x^*(v)$ will be equal to zero. We construct the set V_k by dropping the corresponding vector v from V_{k+1} and by setting $\lambda_k(v) = x^*(v)$. This completes the construction of the subset system.

In the second step, we transform the subset system into the desired permutation. The first d vectors $v_{\pi(1)}, \ldots, v_{\pi(d)}$ are an arbitrary ordering of the vectors in set V_d. For $k \geq d+1$, we choose vector $v_{\pi(k)}$ as the unique element of $V_k - V_{k-1}$. We claim that in the resulting permutation, the norm of every partial sum $\sum_{i=1}^{k} v_{\pi(i)}$ is at most d. Indeed, for $k \leq d$ the triangle inequality together with $\|v_i\| \leq 1$ implies $\|\sum_{i=1}^{k} v_{\pi(i)}\| \leq \sum_{i=1}^{k} \|v_{\pi(i)}\| \leq d$. For

$d+1 \leq k \leq n$, the claim follows from the following chain of equations and inequalities, which is based on properties (A) and (B) and on the triangle inequality.

$$\begin{aligned} \|\sum_{i=1}^{k} v_{\pi(i)}\| &= \|\sum_{v\in V_k} v\| = \|\sum_{v\in V_k} v - \sum_{v\in V_k} \lambda_k(v)\cdot v\| \\ &\leq \sum_{v\in V_k} (1-\lambda_k(v))\cdot\|v\| \leq \sum_{v\in V_k}(1-\lambda_k(v)) \\ &= |V_k| - \sum_{v\in V_k}\lambda_k(v) = k-(k-d) = d \end{aligned}$$

This completes the proof of Theorem 4. Note that the constructed permutation does not depend on the underlying norm. Note furthermore that the entire construction can easily be implemented in polynomial time.

Banaszczyk [2] slightly strengthened the bound in Theorem 4 on the Steinitz constants for norms in d-dimensional space to $c(B) \leq d - 1 + 1/d$. Bergström [5] showed that the Steinitz constant of the Euclidean plane (2-dimensional space with Euclidean norm) equals $\sqrt{5}/2 \approx 1.118$. It is known (and easy to see) that the Steinitz constant of the d-dimensional Euclidean space is at least $\sqrt{d+3}/2$, and this might well be the correct value of the d-dimensional Euclidean Steinitz constant. However, at the current moment not even a sub-linear upper bound is known and the problem is wide open (even for $d = 3$).

5 A tractable special case

Recall that $o_{\max}$ denotes the length of the longest operation, that $p_{\max}$ denotes the length of the longest job, and that $L_{\max}$ denotes the maximum machine load. Throughout this section we will assume that

$$L_1 = L_2 = L_3 = \cdots = L_m = L_{\max} \qquad \text{and} \qquad o_{\max} = 1. \tag{6}$$

The equality of all machine loads in (6) can be reached by adding dummy jobs, and $o_{\max} = 1$ can be reached by scaling. We will apply the machinery for vector rearrangements (as described in the preceding section) to the open shop scheduling problem $Om||C_{\max}$. We introduce a unit ball B^* for a norm $\|\cdot\|_*$ in $(m-1)$-dimensional space, defined by

$$B^* = \left\{(x_1,\ldots,x_{m-1}) \in \mathbb{R}^{m-1} : \ |x_k| \leq 1 \text{ and } |x_k - x_\ell| \leq 1 \text{ for all } k \text{ and } \ell\right\}. \tag{7}$$

Every job J_j with processing times $p_{i,j}$ is translated into an $(m-1)$-dimensional vector

$$v_j = (p_{1,j} - p_{m,j},\ p_{2,j} - p_{m,j},\ \ldots,\ p_{m-1,j} - p_{m,j}). \tag{8}$$

Because of (6) we have $\sum_{j=1}^{n} v_j = 0$ and $\|v_j\|_* \leq 1$ for $1 \leq j \leq n$, so that the conditions (2) for the vector rearrangement Theorem 4 are satisfied. Consequently there exists a permutation $v_{\pi(1)},\ldots,v_{\pi(n)}$ of these vectors, so that

$$\|v_{\pi(1)} + v_{\pi(2)} + \cdots + v_{\pi(k)}\|_* \leq m-1 \qquad \text{for } k = 1,\ldots,n. \tag{9}$$

We construct an infeasible schedule that on each machine processes the jobs without idle time in the ordering $J_{\pi(1)},\ldots,J_{\pi(n)}$; see Figure 2 for an illustration. This schedule is extremely infeasible, as it schedules all operations of every job into a short time interval; this is a consequence of (9) and the definition of norm $\|\cdot\|_*$. The positive effect of this type of

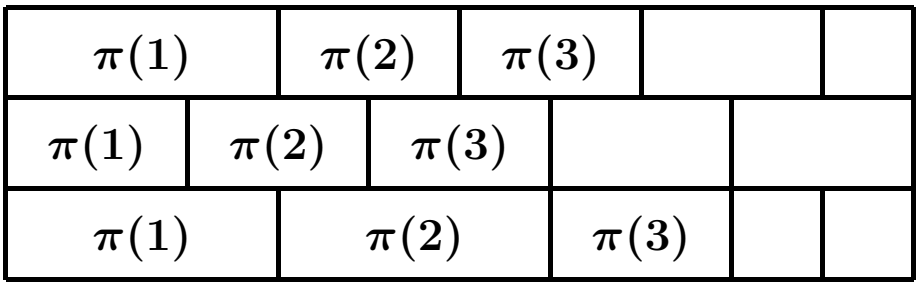


Figure 2 The infeasible schedule that results from the vector rearrangement.

infeasibility is that we have a good understanding of the global structure of this schedule. The completion time of operation $O_{i,j}$ in the infeasible schedule is denoted by $C_{i,j}$. Then for $k \geq 2$ we have

$$\begin{aligned} C_{1,\pi(k)} - C_{2,\pi(k-1)} &= \sum_{j=1}^{k} p_{1,\pi(j)} - \sum_{j=1}^{k-1} p_{2,\pi(j)} \\ &= \sum_{j=1}^{k-1} (p_{1,\pi(j)} - p_{2,\pi(j)}) + p_{1,\pi(k)} \;\leq\; (d-1)+1 \;=\; d. \end{aligned}$$

In the inequality, we use (9) and $p_{1,\pi(k)} \leq o_{\max} = 1$ from (6). By applying analogous arguments, we derive

$$\begin{aligned} \Delta_1 &:= \max_{k\geq 2} C_{m,\pi(k)} - C_{1,\pi(k-1)} &&\leq m \\ \Delta_2 &:= \max_{k\geq 2} C_{1,\pi(k)} - C_{2,\pi(k-1)} &&\leq m \\ \Delta_3 &:= \max_{k\geq 2} C_{2,\pi(k)} - C_{3,\pi(k-1)} &&\leq m \\ &\quad \ldots \quad \ldots \quad \ldots \\ \Delta_m &:= \max_{k\geq 2} C_{m-1,\pi(k)} - C_{m,\pi(k-1)} &&\leq m \end{aligned}$$

This means that we can turn the infeasible schedule into a feasible schedule, by simply shifting all operations on every machine M_i by $(i-1)m$ time units into the future. The makespan of the resulting schedule will be bounded by $L_{\max} + (m-1)o_{\max}$, which yields a reasonable approximation result. We will describe next how to get an even better result. We wrap the infeasible schedule around a cylinder with circumference $L_{\max}$. Each of the individual machine schedules forms a ring around the cylinder that may be rotated. We freeze the ring for machine M_1, and we mark the starting time of job $J_{\pi(1)}$ as the zero-point.

We rotate the ring for machine M_2 by Δ_2 time units and thereby shift the starting time of each operation by Δ_2 into the future. By doing this, we resolve all collisions between operations on M_1 and operations on M_2: Every job has now disjoint processing intervals on M_1 and M_2. Then we rotate the ring for machine M_2 by $\varepsilon_2 \leq o_{\max}$ additional time units, so that one of the operations on M_2 is started at the marked zero-point. Next, we do a similar rotation of the ring for machine M_3 by $\Delta_2 + \Delta_3 + \varepsilon_2 + \varepsilon_3$ time units, so that all collisions between M_2 and M_3 are resolved and so that one of the operations on M_3 is started at the marked zero-point. And so on. The ring for machine M_i is rotated by $\sum_{k=2}^{i} \Delta_k + \varepsilon_k$ time units, so that all collisions are resolved and so that some operation starts at the zero-point.

In the end, we cut the rings open at the marked zero-point and flatten them into a schedule for the considered open shop instance. If $L_{\max} - \Delta_1$ is larger than the total length of all shifts, the resulting schedule will be feasible: Before the shifting, all operations of job J_j were scheduled very close to each other in time. The first shift puts $O_{1,j}$ and $O_{2,j}$ into disjoint processing intervals, and each of the later shifts puts another operation into a disjoint processing interval. As $L_{\max}$ is sufficiently large, the later operations of job J_j will not be shifted all the way around the cylinder and hence cannot cause collisions with the first operation $O_{1,j}$ of that job. Since $\Delta_i \leq m$ and since $\varepsilon_i \leq o_{\max} \leq 1$, the total length of all shifts is at most $(m-1)(m+1)$, and this should be at most $L_{\max} - \Delta_1 \geq L_{\max} - m$.

▶ **Theorem 5** (Fiala [8]). *If an instance of $Om||C_{\max}$ satisfies $L_{\max} \geq (m^2+m-1)o_{\max}$, then the optimal makespan is $L_{\max}$. Furthermore, an optimal schedule can be computed in polynomial time.*

One consequence of Theorem 5 is that open shop problems in the real world are often easy to solve: If all jobs are drawn from the same distribution and if there is a sufficiently large number of jobs, then the condition $L_{\max} \geq (m^2+m-1)o_{\max}$ in Theorem 5 will hold true and the instances become easy to solve.

Belov & Stolin [4] were the first to apply vector rearrangement methods in the area of scheduling (and they applied them to the flow shop problem). Fiala [8] discovered the nice connection to the open shop problem; he actually proved a much stronger version of Theorem 5 where the factor m^2+m-1 is replaced by $8m' \log_2(m') + 5m'$ where m' is the smallest power of 2 greater or equal to m. Bárány & Fiala [3] improved Fiala's bound by a factor of 2, and Sevastianov [16] improved it down to roughly $(16/3)m \log_2 m$. Sevastianov [17] surveys and summarizes the history of vector rearrangement methods in the area of scheduling.

In the light of the above results, it is natural to ask for the smallest value $\eta(m)$, so that every instance of $Om||C_{\max}$ with $L_{\max} \geq \eta(m)o_{\max}$ automatically satisfies $C^*_{\max} = L_{\max}$.

▶ **Open problem 6.** *Derive good upper and lower bounds on $\eta(m)$ for $m \geq 3$.*

Sevastianov [18] establishes the upper bound $\eta(m) \leq m^2 - 1 + 1/(m-1)$, which is useful for small values of m. [18] also establishes the lower bound $\eta(m) \geq 2m-2$. Here is the bad instance for $m=3$ machines that demonstrates $\eta(3) \geq 4$: There is one job with processing time 1 on each machine. Furthermore, for each machine M_i $(i=1,2,3)$ there are three jobs with processing time $1-\varepsilon$ on M_i and processing time 0 on the other two machines. Then $L_{\max} = 4-3\varepsilon$ and $C^*_{\max} = 4-\varepsilon$.

Sevastianov [18] also shows that $Om||C_{\max}$ remains NP-hard, if it is restricted to instances with $L_{\max}/o_{\max} = \rho$ where $1 < \rho < 2m-3$. It is not clear, what is going on for instances with $2m-3 \leq \rho < \eta(m)$. Perhaps, the instances with $\rho < \eta(m)$ are all NP-hard; in that case $\eta(m)$ would be a threshold at which the complexity jumps from NP-hard to trivial.

▶ **Open problem 7.** *Determine the computational complexity of the restriction of $Om||C_{\max}$ to instances with $L_{\max}/o_{\max} = 2m-2$.*

6 Approximation for an arbitrary number of machines

Here is a simple greedy algorithm for $O||C_{\max}$: Start at time $t=0$, and whenever some machine becomes idle and there is some job available that still needs processing on that machine then assign that job to that machine. Ties are broken arbitrarily. This greedy algorithm was formulated by Bárány & Fiala [3] who attribute it to private communication with Anna Racsmány.

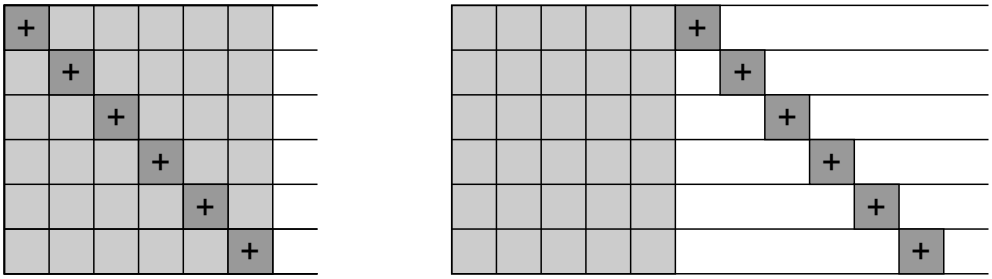


Figure 3 A lower bound instance for the greedy algorithm on $m = 6$ machines. The dummy jobs are shown in light-gray, while the operations of job J^+ are in dark-gray and marked by $+$.

▶ **Theorem 8** (Bárány & Fiala [3]). *The greedy algorithm is a polynomial time approximation algorithm for $O||C_{\max}$ with worst case ratio at most 2.*

Proof. Consider a greedy schedule, and let $O_{i,j}$ be an operation that completes last. Then on machine M_i, the greedy schedule has busy time intervals and idle time intervals. The total length of the busy time intervals is $L_i \leq L_{\max}$. Whenever M_i is idle, it is not processing job J_j and the only possible reason for this is that job J_j is being processed on one of the other machines. Therefore the total length of the idle time intervals is at most $p_j \leq p_{\max}$. This implies that the greedy makespan is at most $L_{\max} + p_{\max}$, which according to (1) is bounded by $2\beta^* \leq 2C^*_{\max}$. ◀

The result in Theorem 8 can also be derived as a corollary to a more general result by Aksjonov [1]. How good is the worst case analysis in this theorem? Consider the following instance with m machines and $m^2 - m + 1$ jobs. For $i = 1, \ldots, m$ there are $m - 1$ dummy jobs that each need one unit of processing time on machine M_i and zero processing time on all other machines. Furthermore, there is a job J^+ that needs one unit of processing time on every machine. There is a greedy schedule with makespan $2m - 1$ for this instance, in which from time 0 to time $m - 1$ all machines are busy with processing the dummy jobs, and from time $m - 1$ to time $2m - 1$ they process job J^+. As the optimal makespan is $C^*_{\max} = m$, the worst case ratio of the greedy algorithm is at least $2 - 1/m$; see Figure 3 for an illustration. Chen & Strusevich [6] have settled the cases $m = 2$ and $m = 3$ of the following open problem by a tedious case analysis, and Chen & Yu [7] have settled the case $m = 4$.

▶ **Open problem 9.** *Prove that the greedy algorithm for $Om||C_{\max}$ has worst case ratio at most $2 - 1/m$.*

A difficult open problem in this area asks whether there is a polynomial time approximation algorithm for $O||C_{\max}$ with worst case ratio strictly better than 2. One possible approach would work with the lower bound β^* defined in (1). Sevastianov & Tchernykh [19] have proved $C^*_{\max} \leq 4\beta^*/3$ for problem $O3||C_{\max}$. Their proof is based on heavy case analysis and on case enumeration with the help of a computer program. As the computer program described in [19] takes more than 200 hours of computation time, this approach does not seem to be applicable to $m \geq 4$ machines.

▶ **Open problem 10.** *Prove that any instance of $Om||C_{\max}$ satisfies $C^*_{\max} \leq 3\beta^*/2$.*

Here is an instance that demonstrates that the factor 3/2 in this open problem would in fact be best possible. The instance uses m machines and $m + 1$ jobs. For $j = 1, \ldots, m$ the job J_j consists of the operation O_{jj} with processing time $p_{jj} = m - 1$ on machine M_j, and with

processing times 0 on the other $m-1$ machines. The final job J_{m+1} has m operations all with processing time 1. Then $\beta^* = m$ and $C^*_{\max} = \lceil m/2 \rceil + m - 1$. As m becomes large, the ratio between $C^*_{\max}$ and β^* tends to $3/2$.

Now let us turn to negative results on the worst case ratio of polynomial time approximation algorithms for $O||C_{\max}$. Williamson & al. [24] prove that it is NP-hard to decide whether an $O||C_{\max}$ instance with integer processing times has optimal makespan at most 4. Since the optimal makespan of a NO-instance is at least 5, a polynomial time approximation algorithm with worst case ratio $5/4 - \varepsilon$ would allow us to distinguish the YES-instances from the NO-instances in polynomial time.

▶ **Theorem 11** (Williamson & al. [24]). *Unless P=NP, problem $O||C_{\max}$ does not allow a polynomial time approximation algorithm with worst case ratio strictly better than $5/4$.*

It might be possible to lift the hardness proof in [24] to get stronger inapproximability results.

▶ **Open problem 12.** *Analyze the computational complexity of the (a, b)-versions of $O||C_{\max}$ instances with integer processing times: Decide whether the optimal makespan does satisfy $C^*_{\max} \leq a$ or whether it does satisfy $C^*_{\max} \geq b$.*

If this (a, b)-version turns out to be NP-hard for fixed integers a and b, then $O||C_{\max}$ cannot have a polynomial time approximation algorithm with worst case ratio strictly better than b/a unless $P = NP$. The result in [24] yields NP-hardness of the $(4, 5)$-version, and [24] also shows that the $(3, 4)$-version is solvable in polynomial time. Hence the smallest interesting open cases would concern the $(5, 7)$-version and the $(6, 8)$-version.

7 Approximation for a fixed number of machines

For an arbitrary number of machines, polynomial time approximation algorithms cannot have worst case ratios very close to 1; see Theorem 11. For a fixed number of machines, the situation is much better and there is a polynomial time approximation scheme (PTAS).

▶ **Theorem 13** (Sevastianov & Woeginger [20]). *For every fixed $m \geq 3$, problem $Om||C_{\max}$ has a PTAS.*

We now show a proof of Theorem 13 for the special case $m = 3$. (The general case is based on exactly the same ideas, while some of the details become a bit messier.) So let us consider some instance of $O3||C_{\max}$, and let ε with $0 < \varepsilon < 1$ be some small real number that indicates the desired precision of approximation. The running time of our algorithm will be polynomial in the instance size, but exponential in $1/\varepsilon$. The resulting makespan will come arbitrarily close to $C^*_{\max}$, if we let ε tend to 0.

As often in approximation schemes for scheduling problems, the jobs are classified into *big* and into *small* jobs. We call a job *big*, if one of its operations has length $p_{i,j} \geq \varepsilon\beta^*$, where β^* is the lower bound defined in (1). All other jobs are *small* jobs, and we want to assume for the moment that (***) all operations of all small jobs have length $p_{i,j} \leq \varepsilon^2\beta^*$; we will show later how to work around this assumption. Since the total processing time of all jobs is at most $3L_{\max} \leq 3\beta^*$ and as every big job has processing time at least $\varepsilon\beta^*$, there are at most $3/\varepsilon$ big jobs. The algorithm now works in two phases.

- In the first phase, we determine an optimal schedule for the big jobs. This can be done in $O(1)$ time, as the running time does only depend on $1/\varepsilon$ and does not depend on the instance size. In the resulting schedule, every machine processes at most $3/\varepsilon$ operations with at most $3/\varepsilon$ gaps of idle time between the operations; see Figure 4 for an illustration.

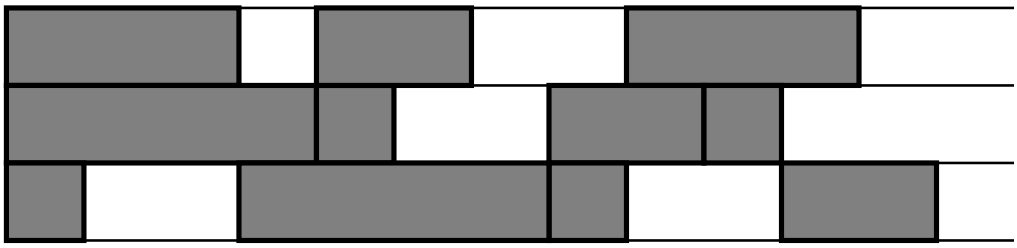


Figure 4 An optimal schedule for the big jobs in the proof of Theorem 13.

- In the second phase, we pack the operations of the small jobs into the idle gaps. This is done greedily (as in Theorem 8). Start at time $t = 0$, and whenever some machine becomes idle at some time t, try to process an unprocessed small operation on that machine. There are two possible scenarios under which an unprocessed small operation $O_{i,j}$ cannot be started at time t: First another operation O_{kj} of the same job might currently be processed on some other machine. Secondly, the remaining part of the current gap might be too small to accommodate $O_{i,j}$. If one of these scenarios occurs, we try to schedule some other small operation. If no operation can be scheduled, then the machine is left idle for the moment.

Now let us analyze the makespan $C^A_{\max}$ of the resulting approximating schedule. Let $O_{i,j}$ be an operation that completes last. In the first case assume that $O_{i,j}$ belongs to a big job. Then $C^A_{\max}$ coincides with the optimal makespan for the big jobs, and we actually have found an optimal schedule. In the second case assume that $O_{i,j}$ belongs to a small job. Then we consider the busy time intervals and the idle time intervals on machine M_i. The total length of all busy time intervals is the load $L_i \leq \beta^*$. Whenever machine M_i was idle, it could not process operation $O_{i,j}$. This means that either (i) job J_j was being processed on one of the other machines, or that (ii) the remaining gap was too small to accommodate $O_{i,j}$. The total idle time of type (i) is bounded by the length of the small job J_j, which is at most $3\varepsilon^2\beta^*$. The total idle time of type (ii) is bounded by the number of gaps multiplied by the length of operation $O_{i,j}$, which is at most $(3/\varepsilon) \cdot (\varepsilon^2\beta^*) = 3\varepsilon\beta^*$. Altogether, this implies that the approximating makespan can be bounded as

$$C^A_{\max} \;=\; \text{Busy} + \text{Idle(i)} + \text{Idle(ii)} \;\leq\; \beta^* + 3\varepsilon^2\beta^* + 3\varepsilon\beta^* \;\leq\; (1 + 3\varepsilon^2 + 3\varepsilon)\, C^*_{\max}. \qquad (10)$$

As ε tends to 0, the error factor $1 + 3\varepsilon^2 + 3\varepsilon$ tends to 1. This yields the desired PTAS modulo the assumption (***).

It remains to discuss what to do with assumption (***), which essentially postulates an empty no man's land between big operations (of length at least $\varepsilon\beta^*$) and small operations (of length at most $\varepsilon^2\beta^*$). In other words, under assumption (***) non-big jobs must not contain operations of intermediate length $\varepsilon^2\beta^* < p_{i,j} < \varepsilon\beta^*$. This assumption will be totally wrong for most instances, but we can come very close to it by playing around with the value of ε. This is done as follows.

For a real number δ with $0 < \delta < 1$, we say that a job is δ-*big*, if one of its operations has length $p_{i,j} \geq \delta\beta^*$ and otherwise it is δ-*small*. An operation $O_{i,j}$ is δ-*nasty*, if it belongs to a δ-small job and satisfies the inequality $\delta^2\beta^* < p_{i,j} < \delta\beta^*$. By $N(\delta)$ we denote the total length of all δ-nasty operations. Now consider the real numbers $\delta_k = \varepsilon^{2^k}$ for $k \geq 0$. Then every operation $O_{i,j}$ is δ_k-nasty for at most one choice of index k. We search for an index k that satisfies the inequality

$$N(\delta_k) \;\leq\; \varepsilon\beta^*. \qquad (11)$$

If some δ_k violates (11), then the corresponding δ_k-nasty operations consume at least $\varepsilon\beta^*$ of the total processing time of all operations (which is at most $3\beta^*$). Hence some $k \leq 3/\varepsilon$ will indeed satisfy (11). From now on we work with that particular index k and with that particular value δ_k.

The final approximation scheme works as follows. First we remove from the instance all the δ_k-small jobs that contain some δ_k-nasty operation. To the surviving jobs we apply the original approximation algorithm as described above with $\varepsilon := \delta_k$, and thereby find a schedule with makespan at most $(1 + 3\delta_k^2 + 3\delta_k)\, C^*_{\max}$ according to (10). In the end, we greedily add the previously removed jobs with δ_k-nasty operations to this schedule. Since the overall processing time of all removed jobs is at most $3\varepsilon\beta^*$, this increases the makespan by at most $3\varepsilon\beta^*$. Since $\delta_k \leq \varepsilon$, this altogether yields a schedule of makespan at most $(1 + 3\varepsilon^2 + 6\varepsilon)C^*_{\max}$. This completes the proof of Theorem 13 for the special case $m = 3$.

An FPTAS (fully polynomial time approximation scheme) is a PTAS whose time complexity is also polynomially bounded in $1/\varepsilon$. The following open problem is closely linked to the existence of pseudo-polynomial time exact algorithms for $Om||C_{\max}$.

▶ **Open problem 14.** *Prove that problem $Om||C_{\max}$ has an FPTAS for every fixed $m \geq 3$.*

References

1. V.A. Aksjonov. A polynomial-time algorithm for an approximate solution of a scheduling problem (in Russian). *Upravlyaemye Sistemy*, 28:8–11, 1988.
2. W. Banaszczyk. The Steinitz constant of the plane. *Journal für die Reine und Angewandte Mathematik*, 373:218–220, 1987.
3. I. Bárány and T. Fiala. Nearly optimum solution of multimachine scheduling problems (in Hungarian). *Szigma Mathematika Közgazdasági Folyóirat*, 15:177–191, 1982.
4. I.S. Belov and Ya. N. Stolin. An algorithm for the single-route scheduling problem (in Russian). In *Matematiceskaja ékonomika i funkcionalnyi analiz (Mathematical Economics and Functional Analysis)*, pages 248–257. Nauka, Moscow, 1974.
5. V. Bergström. Zwei Sätze über ebene Vektorpolygone. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 8:148–152, 1931.
6. B. Chen and V.A. Strusevich. Approximation algorithms for three-machine open shop scheduling. *ORSA Journal on Computing*, 5:321–326, 1993.
7. B. Chen and W. Yu. How good is a dense shop schedule? *Acta Mathematicae Applicatae Sinica*, 17:121–128, 2001.
8. T. Fiala. An algorithm for the open-shop problem. *Mathematics of Operations Research*, 8:100–109, 1983.
9. M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
10. T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the Association for Computing Machinery*, 25:92–101, 1976.
11. V. Grinberg and S.V. Sevastianov. The value of the Steinitz constant (in Russian). *Funktionalnyi Analiz i ego Prilozheniia (Functional Analysis and Its Applications)*, 14:125–126, 1980.
12. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin, editors, *Logistics of Production and Inventory (Handbooks in Operations Research and Management Science)*, pages 445–522. North-Holland, Amsterdam, 1993.
13. J.K. Lenstra. Strong NP-hardness of open shop scheduling. Unpublished manuscript, 1978.
14. S. Sahni and Y. Cho. Complexity of scheduling shops with no wait in process. *Mathematics of Operations Researc*, 4:448–457, 1979.

15 S.V. Sevastianov. Approximate solution of some problems in scheduling theory (in Russian). *Metody Diskretnogo Analiza*, 32:66–75, 1978.

16 S.V. Sevastianov. A polynomially solvable case of the open shop problem with arbitrary number of machines (in Russian). *Kibernetika i Systemnii Analiz*, 6:135–154, 1992.

17 S.V. Sevastianov. On some geometric methods in scheduling theory: a survey. *Discrete Applied Mathematics*, 55:59–82, 1994.

18 S.V. Sevastianov. Vector summation in Banach space and polynomial algorithms for flow shops and open shops. *Mathematics of Operations Research*, 20:90–103, 1995.

19 S.V. Sevastianov and I.D. Tchernykh. Computer-aided way to prove theorems in scheduling. In *Proceedings of the 6th European Symposium on Algorithms (ESA'1998)*, LNCS 1461, pages 502–513. Springer, 1998.

20 S.V. Sevastianov and G.J. Woeginger. Makespan minimization in open shops: A polynomial time approximation scheme. *Mathematical Programming*, 82:191–198, 1998.

21 N.V. Shaklevich and V.A. Strusevich. Two machine open shop scheduling problem to minimize an arbitrary machine usage regular penalty function. *European Journal of Operational Research*, 70:391–404, 1993.

22 E. Steinitz. Bedingt konvergente Reihen und konvexe Systeme. *Journal für die Reine und Angewandte Mathematik*, 143:128–176, 1913.

23 M. van den Akker, J.A. Hoogeveen, and G.J. Woeginger. The two-machine open shop problem: to fit or not to fit, that is the question. *Operations Research Letters*, 31:219–224, 2003.

24 D.P. Williamson, L.A. Hall, J.A. Hoogeveen, C.A.J. Hurkens, J.K. Lenstra, S.V. Sevastianov, and D.B. Shmoys. Short shop schedules. *Operations Research*, 45:288–294, 1997.

Approximating Airports and Railways

Anna Adamaszek
University of Copenhagen, Copenhagen, Denmark
anad@di.ku.dk

Antonios Antoniadis
Saarland University and MPII, Saarbrücken, Germany
aantonia@mpi-inf.mpg.de

Amit Kumar
IIT Delhi, Delhi
amitk@cse.iitd.ac.in

Tobias Mömke
Saarland University, Saarbrücken, Germany and
University of Bremen, Bremen, Germany
moemke@cs.uni-saarland.de

Abstract

We consider the airport and railway problem (AR), which combines capacitated facility location with network design, both in the general metric and the two-dimensional Euclidean space. An instance of the airport and railway problem consists of a set of points in the corresponding metric, together with a non-negative weight for each point, and a parameter k. The points represent cities, the weights denote costs of opening an airport in the corresponding city, and the parameter k is a maximum capacity of an airport. The goal is to construct a minimum cost network of airports and railways connecting all the cities, where railways correspond to edges connecting pairs of points, and the cost of a railway is equal to the distance between the corresponding points. The network is partitioned into components, where each component contains an open airport, and spans at most k cities. For the Euclidean case, any points in the plane can be used as Steiner vertices of the network.

We obtain the first bicriteria approximation algorithm for AR for the general metric case, which yields a 4-approximate solution with a resource augmentation of the airport capacity k by a factor of 2. More generally, for any parameter $0 < p \leq 1$ where $p \cdot k$ is an integer we develop a $(4/3)(2 + 1/p)$-approximation algorithm for metric AR with a resource augmentation by a factor of $1 + p$.

Furthermore, we obtain the first constant factor approximation algorithm that does not resort to resource augmentation for AR in the Euclidean plane. Additionally, for the Euclidean setting we provide a quasi-polynomial time approximation scheme for the same problem with a resource augmentation by a factor of $1 + \mu$ on the airport capacity, for any fixed $\mu > 0$.

2012 ACM Subject Classification Mathematics of computing → Approximation algorithms, Theory of computation → Routing and network design problems, Theory of computation → Facility location and clustering

Keywords and phrases Approximation Algorithms, Network Design, Facility Location, Airports and Railways

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.5

Funding Research supported in part by the Danish Council for Independent Research DFF-MOBILEX mobility grant and by Deutsche Forschungsgemeinschaft grants AN 1262/1-1 and MO 2889/1-1.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 5; pp. 5:1–5:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

This paper studies the airport and railway problem, which combines facility location and network design, and has been introduced Adamaszek et al. [1]. In the airport and railway problem the input consists of a complete n-vertex graph G with vertex costs $a\colon V(G) \to \mathbb{R}_{\geq 0}$ and edge costs $\text{len}\colon E(G) \to \mathbb{R}_{\geq 0}$, and a parameter k. The vertices represent cities, vertex cost represents the cost of opening an airport in the corresponding city, and edge cost models the cost of building a railway connecting the corresponding pair of cities. Finally, the parameter k represents a maximum capacity of an airport. The goal is to compute a minimum cost network of airports $A \subseteq V(G)$ and railways $R \subseteq E(G)$ which satisfies the following properties: (i) each $v \in V(G)$ is connected with some vertex from A via a path of edges from R (i.e., all cities are connected by the network), and (ii) each connected component of the network contains at most k vertices (i.e., each airport serves at most k cities). The cost of the network equals $a(A) + \text{len}(R) = \sum_{v \in A} a(v) + \sum_{e \in R} \text{len}(e)$. As the cost functions a and len are non-negative, an optimal solution to AR is a forest, with the cheapest airport opened in each connected component.

We consider both the case where $(V(G), \text{len})$ is a general metric space, and the case where it is the Euclidean plane, i.e., the set of vertices $V(G)$ is represented by a set of points in the Euclidean plane, and the edge cost len is the Euclidean distance between the corresponding points. The goal in both cases is to find a minimum cost network spanning all vertices $V(G)$ and consisting of *components*, such that each component spans at most k vertices and contains an open airport. Furthermore, for the Euclidean metric case, we assume that each point in the Euclidean plane can be used as a Steiner vertex within the components. Note that in the Euclidean plane we allow edges corresponding to different components to cross, without a Steiner vertex at the intersection.

We also consider AR with resource augmentation, denoted by AR_α for a constant $\alpha > 1$, where we are allowed to assign $\alpha \cdot k$ cities to an airport of capacity k. We then compare the cost of the obtained solution against an optimal solution without resource augmentation.

Related work. The airport and railway problem AR is the most general problem within the framework introduced by Adamaszek et al. [1]. Several interesting novel problems can be defined within this framework by starting with AR and imposing additional constraints to the underlying network. It was shown in [1] that two-dimensional Euclidean AR is NP-hard, even when all the vertex costs are uniform. This uniform-vertex-cost case admits a polynomial time approximation scheme. Furthermore, when the airport capacity k is unbounded, AR can be solved exactly in polynomial time, even with both arbitrary vertex costs and arbitrary edge costs. Additionally, [1] considered the related AR_P problem. In AR_P, each component of the network is required to be a path, with airports at both of its endpoints. This problem is of particular interest because it models the Capacitated Vehicle Routing Problem (CVRP). Two-dimensional Euclidean AR_P is shown to be NP-hard even with uniform airport costs and unbounded parameter k. For the setting where either the airport costs are uniform or the parameter k is unbounded, a PTAS for AR_P has been presented.

Since AR combines the classical capacitated facility location (CFL) problem and network design (ND), we shortly describe these problems.

- **Capacitated Facility Location (CFL):** We are given a complete graph G with $V(G) := \{v_1, \ldots, v_n\}$, edge costs $d\colon E(G) \to \mathbb{R}_{\geq 0}$ and vertex costs $c\colon V(G) \to \mathbb{R}_{\geq 0}$, along with a capacity parameter k. Intuitively, $d(v_i, v_j)$ denotes the distance between vertices v_i and v_j, and $c(v_i)$ denotes the cost of opening a facility at v_i. A feasible solution to CFL

consists of a set of open facilities $F \subseteq V(G)$, and an assignment of each vertex v_i to some open facility $f(v_i) \in F$ so that each facility has at most k vertices assigned. The cost of a solution is given by the sum of the cost for opening the facilities and the cost of connecting all vertices to the assigned facilities by *direct links*, i.e., $\sum_{v \in F} c(v) + \sum_{v \in V(G)} d(v, f(v))$. The goal is to find a minimum cost feasible solution. For CFL, the currently best results are a 1.488-approximation algorithm [8] and an LP based constant factor algorithm [4].

- **Network Design (ND):** In the framework of network design we are given a graph G with weights on the edges, and in some cases also on the vertices. Furthermore, we are given a set of constraints, e.g., connectivity constraints. The goal is to find a set of edges of minimum cost that satisfy the constraints.

Another problem closely related to AR is the *capacitated minimum spanning tree problem* (CMST). In CMST, the goal is to construct a minimum cost collection of trees covering all the input vertices, each tree spanning at most k vertices, connected to a single pre-specified root. Jothi and Raghavachari [7] give a 3.15-approximation algorithm for Euclidean CMST and a $2 + \gamma$ approximation for metric CMST, where $\gamma \leq 2$ is the ratio between the cost of a Steiner tree and a minimum spanning tree. Both results allow demands on the vertices. We note that the AR problem can be modelled as CMST with an arbitrary (i.e., non-metric) cost function[1]. However, to the best of our knowledge, such version of CMST has not been studied before.

Ravi and Sinha [10] consider a related *capacitated-cable facility location problem* (CCFL) obtaining a constant factor approximation algorithm. The problem is based on the uncapacitated facility location (UFL) problem, i.e., there is a set of facilities with unbounded capacities and non-negative opening costs. But instead of connecting clients to facilities by direct links, they are connected by a network of capacitated cables (i.e., each edge of the constructed network can accommodate at most u units of flow from the clients to the facilities, where u is a parameter). When the cable capacity u is 1, the problem is equivalent to UFL. When the cable capacity $u = \infty$, CCFL is equivalent to AR with $k = \infty$. In general, the problem differs considerably from AR, as in CCFL, once a facility has been opened, it can receive an unbounded amount of flow. CCFL resembles AR, when instead of a bound on the airport capacity we have a bound on the railroad capacity.

Another problem related to AR and CCFL is *capacitated geometric network design* (CGND), where the goal is to create a network of capacitated links which allows sending flow from all the vertices to a single, pre-specified sink. In CGND the optimal network can be more complicated than a tree. Adamaszek et al. [2] provide a PTAS for the two-dimensional Euclidean CGND for link capacities $k \leq 2^{O(\sqrt{\log n})}$, where the network can use Steiner vertices anywhere in the plane.

Maßberg and Vygen [9] obtained a 4.1-approximation for another problem related to AR with uniform airport costs, called the *sink clustering problem*. They construct a network consisting of components, where instead of a capacity constraint for each component, they have a different constraint which incorporates both the capacity and the length of the edges of the component.

[1] To model an instance (G, a, r, k) of AR as a CMST problem, we proceed as follows. We extend the graph G to G' by adding a new vertex v and connecting it with all other vertices of G'. We extend the edge cost function r to a function r' as follows. Each edge $\{u, v\}$ for $u \in V(G)$ has cost equal to $a(u)$ in G'. Then (G', r', v, k) is an instance of CMST, with a pre-specified root v and parameter k. The corresponding instances of AR and CMST have corresponding solutions of the same costs, where adding an edge $\{u, v\}$ to a solution for CMST corresponding to opening an airport at u in AR.

Our results. We initiate the study of AR for general metric spaces from the perspective of bicriteria approximation, where we allow resource augmentation for the airport capacity parameter k. We prove the following theorem, which is the first result for the airport and railway problem on general metric spaces.

▶ **Theorem 1.** *There is a 4-approximation algorithm for metric* AR_2*. More general, for any* $0 < p \leq 1$ *such that* $p \cdot k$ *is integer, there is a* $\frac{4}{3}(2 + \frac{1}{p})$*-approximation algorithm for metric* AR_{1+p}*.*

The algorithm starts with computing an infeasible solution to the problem, returned by an algorithm for uncapacitated AR (i.e., assuming $k = \infty$). Then, this infeasible solution is transformed into a feasible one in a sequence of (technically involved) steps. First, the connected components of the uncapacitated solution are partitioned into paths, where each path contains $k \cdot p$ cities, plus one shorter path per component of the uncapacitated solution. These shorter paths get connected to the airports open by the uncapacitated solution, and they are the reason for the required resource augmentation. Then, the paths containing $k \cdot p$ cities each are assigned to airports using min-cost max-flow computation in a specially constructed graph, where each airport gets connected to at most $1/p$ paths. This requires the solution to open additional airports, and the solution has to be modified again so that each component contains one airport.

We then turn our attention to AR in the Euclidean plane, providing the first approximation algorithm for this setting. Note that this algorithm, in contrast to the algorithm from Theorem 1, does not use resource augmentation.

▶ **Theorem 2.** *For any fixed* $\epsilon > 0$ *there is a* $(2 + \frac{k}{k-1} + \epsilon)$*-approximation algorithm for* AR *with the airport capacity* $k \geq 2$ *in the Euclidean plane.*

Note that for $k = 1$ the AR problem becomes trivial (as the solution requires opening airports in all the cities). For $k \geq 2$, the approximation factor of the algorithm from Theorem 2 is at most $4 + \epsilon$.

In order to obtain Theorem 2, we define a relaxed version AR' of the AR problem, where each component can contain multiple airports and multiple copies of the same edge, each component allows routing flow from all its cities to the airports, each airport serves at most k cities, and each copy of an edge can be used by at most k cities. Note that in this version of the problem the cities belonging to different airports can share the edges of the network. Building upon Arora's PTAS for the Euclidean TSP [5] we develop $(1 + \epsilon)$-approximation algorithm for AR' for any fixed $\epsilon > 0$. By applying a carefully-designed sequence of transformations on the solution returned by the algorithm for AR', we transform it to a feasible solution for AR. These steps resemble the steps of the algorithm from Theorem 1. However, we have to be more careful to avoid resource augmentation.

Finally, we provide a QPTAS for $\mathrm{AR}_{1+\mu}$ for any fixed $\mu > 0$, matching the corresponding result for capacitated facility location [6].

▶ **Theorem 3.** *For arbitrary* $\epsilon, \mu > 0$ *there is a* $(1 + \epsilon)$*-approximation algorithm for two-dimensional Euclidean* $\mathrm{AR}_{1+\mu}$ *running in quasipolynomial time.*

In Section 2 we study metric AR and prove Theorem 1. In Section 3 we study AR in the Euclidean plane. We prove Theorem 3 in Section 3.1 and Theorem 2 in Section 3.3.

2 The Metric Case

In this section we will assume that the edge cost len is metric, i.e., it satisfies the triangle inequality $(\mathrm{len}(\{u, v\}) + \mathrm{len}(\{v, w\}) \geq \mathrm{len}(\{u, w\})$ for each triple of vertices $u, v, w \in V(G))$.

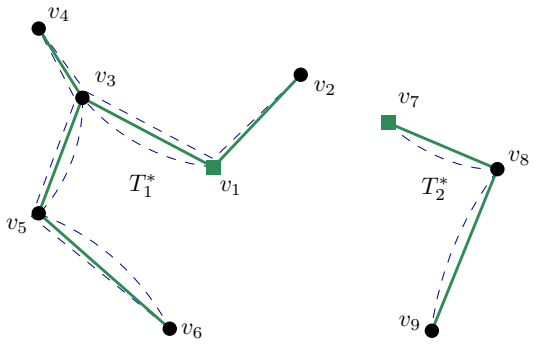


Figure 1 An infeasible solution S_0 for an instance I of AR for $k = 2$. The solution consists of the set of trees $\{T_1^\star, T_2^\star\}$, and airports open at vertices v_1 and v_7. The airports are denoted by squares. The dashed blue lines show an Eulerian tour needed in Step 1.

Fix a parameter $0 < p \leq 1$ to determine the amount of resource augmentation. We assume that p is a multiple of $1/k$, so that $p \cdot k$ is an integer.

Consider an instance I of AR, and let OPT be an optimal solution for I. Our algorithm consists of several steps, which we describe below.

Step 0: Preprocessing. Infeasible solution S_0. We create a new problem instance I_0 by taking I and setting the airport capacity $k = \infty$. I_0 is an instance of AR_F^∞, a relaxation of AR defined in [1], where we assume that the airport capacity is unbounded. By Theorem 4 in [1], there is a polynomial time algorithm for AR_F^∞. The algorithm is an extension of an algorithm finding a minimum spanning tree, and it always outputs a spanning forest.

Let S_0 be an optimal solution for I_0 output by the algorithm from [1]. Note that S_0 may contain components with more than k cities, and therefore it is not necessarily a feasible solution for AR. See an example in Figure 1. In the following lemma we prove various properties of S_0.

▶ **Lemma 4.** *The solution S_0 has the properties that (i) it is a forest, (ii) each connected component of S_0 contains exactly one airport, and (iii) $\texttt{cost}(S_0) \leq opt$.*

▶ **Observation 1.** *If for some instance I, the solution S_0 returned by the AR_F^∞ algorithm only contains components of size at most k, then by Lemma 4 S_0 is already optimal for AR on I. However, in general S_0 may contain components with more than k vertices – in which case it is not a feasible solution for AR.*

Step 1: Splitting each component of S_0 into paths. Infeasible solution S_1. We will split each connected component of S_0 into paths. Each path, except exactly one shorter path, will contain exactly $p \cdot k$ cities (vertices from $V(G)$). We proceed as follows.

By Lemma 4, the edges of S_0 form a forest $\{T_1^\star, \ldots, T_\ell^\star\}$ in G, where each tree $T_i^\star$ of S_0 contains one airport. For each tree $T_i^\star$ of S_0, denote by v_i the vertex of $T_i^\star$ with an airport, and consider $T_i^\star$ as a rooted tree with a root at v_i. Observe that $\texttt{cost}(S_0) = \sum_{i=1,\ldots,\ell} \texttt{cost}(T_i^\star)$, where $\texttt{cost}(T_i^\star) = a(v_i) + \sum_{e \in E(G) \cap T_i^\star} \text{len}(e)$ denotes the total cost of the tree $T_i^\star$.

First, in the solution S_1 we open the airport at v_i for each tree $T_i^\star$. Recall that S_0 also opens (and therefore pays for) these airports. The next step is transforming the forest $\{T_1^\star, \ldots, T_\ell^\star\}$ into a collection of paths.

For each tree $T_i^\star$ we construct a path $P_i^\star$ starting in the vertex v_i, visiting all cities of $T_i^\star$, and such that the cost of edges of $P_i^\star$ is at most twice the cost of the edges of $T_i^\star$. We

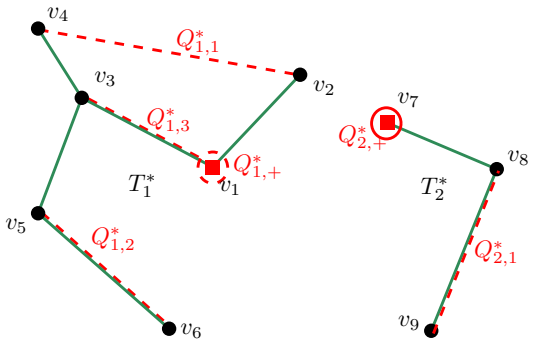


Figure 2 An infeasible solution S_1 for the instance I of AR from Figure 1 (green) is pictured in red (dashed). Here $k = 2$ and $p = 1$, i.e., each subpath $Q^\star_{i,j}$ contains 2 cities, and each subpath $Q^\star_{i,+}$ contains either 0 or 1 cities. $Q^\star_{1,+}$ is empty, incident with the vertex v_1, and the airport v_1 does not serve any city. $Q^\star_{2,+}$ consists of a city v_7, and the airport v_7 serves this city.

do that by doubling all edges of $T^\star_i$, constructing an Eulerian tour of $T^\star_i$ (note that after doubling the edges each vertex has even degree), shortcutting the tour so that each vertex is visited only once, and removing from the obtained tour one edge incident with v_i.

We break each of the paths $P^\star_i$ into subpaths $Q^\star_{i,j}$, each containing exactly $p \cdot k$ cities, and exactly one shorter subpath $Q^\star_{i,+}$ whose number of cities lies in the range $[0, p \cdot k - 1]$. We do this so that the subpath $Q^\star_{i,+}$ is the one closest to the root v_i. In particular, if $Q^\star_{i,+}$ contains at least one city, then it contains the city v_i. If $Q^\star_{i,+}$ is an empty path, we think of it as a path consisting of a single vertex v_i, but not containing the city at v_i. We add all the edges of all the paths $Q^\star_{i,j}$ and $Q^\star_{i,+}$ to S_1. See Figure 2 for an example of this construction.

For each airport v_i, we *assign* the subpath $Q^\star_{i,+}$ to v_i. This means that every city in the (possibly empty) subpath $Q^\star_{i,+}$ is *served* by the airport v_i. Note that this can be done, as by the construction of S_1 all vertices of $Q^\star_{i,+}$ are in the same connected component of S_1 as v_i. [2]

In the subsequent step, we will initially ignore the already assigned cities from the subpaths $Q^\star_{i,+}$, and concentrate on assigning the subpaths $Q^\star_{i,j}$ to airports. The already assigned cities from the subpaths $Q^\star_{i,+}$ will later be added, and they will induce a resource augmentation of the airport capacities.

▶ **Lemma 5.** *The solution S_1 satisfies that (i) each airport serves at most $k \cdot p - 1$ cities*[3]*, (ii) each city from a subpath $Q^\star_{i,+}$ is served by an airport, and (iii)* $\texttt{cost}(S_1) \leq 2 \cdot \texttt{cost}(S_0) \leq 2 \cdot opt$.

▶ **Observation 2.** *If for some instance I every component of S_1 has an airport that serves all cities of the component (including itself), then S_1 is a feasible, 2-approximate solution for I. Such a solution does not use resource augmentation, and each airport is only used up to a capacity of $pk - 1$. However, in general S_1 may have components of size exactly $k \cdot p$ whose cities are not served by any airport, and it is therefore in general not a feasible solution for* AR.

[2] Note that in case when $Q^\star_{i,+}$ is an empty path, the connected component of S_1 containing v_i consists of some path $Q^\star_{i,j}$. However, we do not assign the subpath $Q^\star_{i,j}$ to the vertex v_i, i.e., for now we treat the cities of $Q^\star_{i,j}$ as not served by any airport. Later the solution will be modified, and the cities from $Q^\star_{i,j}$ will be served by some airport (possibly different from v_i).

[3] Recall that in the case when $Q^\star_{i,+}$ is empty, the airport v_i does not serve any city, and the connected component of S_1 containing v_i contains $k \cdot p$ unserved cities.

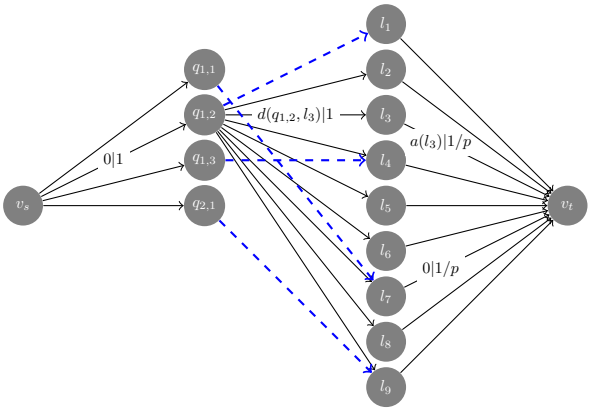


Figure 3 Example of an instance of the min-cost max-flow problem, corresponding to the AR instance from Figures 1 and 2. The goal is to send flow from v_s to v_t. Here $Q = \{q_{1,1}, q_{1,2}, q_{1,3}, q_{2,1}\}$ and $L = \{l_1, \ldots, l_9\}$. An label $x|y$ denotes an edge with cost x and capacity y. The airport l_7 has already been opened in S_1 (and therefore the edge $\{l_7, v_t\}$ has cost 0), while the airport l_3 has not been opened in S_1 (and the edge $\{l_3, v_t\}$ has non-zero cost). Note that for the clarity of presentation not all edges of E_{QL} have been drawn. The blue dashed edges denote a potential solution, i.e., a potential assignment of subpaths to airports.

Step 2: Assigning the subpaths $Q^\star_{i,j}$ to airports using network flows. Infeasible solution S_2. In this step we will make sure that every connected component is assigned to a neighboring airport, and therefore all the cities are served. For that, we will choose a set of additional airports to be opened. We will assign (and connect by choosing the appropriate edges of G) at most $1/p$ many subpaths $Q^\star_{i,j}$ to each airport, considering both the newly opened airports and the airports opened in Step 1. Note that if $1/p$ subpaths get assigned in this step to an airport that has been opened in Step 1 (and therefore might already be serving some cities), the capacity of the airport can become violated. Therefore, the solution S_2 constructed in this step requires resource augmentation. For now, we allow assigning subpaths $Q^\star_{i,j}$ to airports from different components. We will fix that in the subsequent step.

To decide which additional airports should be opened, and how we should assign (and connect) the subpaths $Q^\star_{i,j}$ to the airports, we use min-cost max-flow computation. In this computation we ignore the subpaths $Q^\star_{i,+}$ containing less than $k \cdot p$ cities each, as they have already been assigned (and connected) to the airports.

We construct a directed graph G', with capacities and a cost function d on the edges, as follows (see Figure 3). We introduce a vertex $q_{i,j}$ corresponding to each subpath $Q^\star_{i,j}$ (but not for the subpaths $Q^\star_{i,+}$), and we denote this set of vertices by Q. We also introduce a vertex l_v for each vertex $v \in V(G)$ of the original instance, and we denote this set of vertices by L. For each pair of vertices $(q_{i,j}, l_v) \in Q \times L$ we introduce an edge with capacity 1 and cost $d(q_{i,j}, l_v) := \min_{u \in V(Q^\star_{i,j})} \mathrm{len}(\{u, v\})$, directed from $q_{i,j}$ to l_v. Note that $d(q_{i,j}, l_v)$ equals the minimum distance between a vertex of the subpath $Q^\star_{i,j}$ represented by $q_{i,j}$, and the vertex v corresponding to l_v. We denote this set of edges by E_{QL}. Intuitively, sending flow 1 through an edge $\{q_{i,j}, l_v\} \in E_{QL}$ means connecting the subpath $Q^\star_{i,j}$ to an airport at the vertex v.

We then introduce a source vertex v_s and directed edges from v_s to all vertices in Q, each edge $\{v_s, q_{i,j}\}$ with capacity 1 and cost $d(v_s, q_{i,j}) = 0$. Finally, we introduce a sink vertex v_t and directed edges from each vertex $l_v \in L$ to v_t. We denote these sets of edges by E_Q and E_L, respectively. The cost $d(l_v, v_t)$ of an edge $\{l_v, v_t\}$ is zero if an airport at v has been

opened in S_1 (i.e., in Step 1 of the algorithm), and $a(v)$ otherwise. Each edge of E_L has a capacity of $1/p$, which enforces that no more than $1/p$ subpaths (and therefore no more than $(k \cdot p) \cdot (1/p) = k$ vertices) are assigned to each airport at this step of the algorithm.

The graph G', together with the edge capacities and edge costs, yields an instance of the min-cost max-flow problem, where we want to send flow from the source vertex v_s to the sink vertex v_t. Clearly, the instance admits a solution where the amount of flow is $|Q|$. We can send one unit of flow from v_s to each of the vertices $q_{i,j}$, then also one unit of flow from each $q_{i,j} \in Q$ to some vertex $l_v \in L$ so that $v \in Q^\star_{i,j}$, and as each vertex from L gets at most one unit of flow, it can be sent to the sink vertex v_t. We note that this is the maximum amount of flow that can be sent, since the total capacity of the outgoing edges from v_s is $|Q|$.

It is well known that one can find an optimal, integral solution for the min-cost max-flow problem in time polynomial in the number of vertices and edges of the input instance (cf. [3], Chapters 9 and 10 or [11]). Let S' be this optimal, integral min-cost max-flow problem solution for G'. We denote the cost of S' by $\texttt{cost}(S')$.

▶ **Lemma 6.** *The following inequality holds:* $\texttt{cost}(S') \leq opt/p$.

Proof. We will show how we can translate OPT into a solution OPT$'$ to the min-cost max-flow problem, with cost of at most opt$/p$. By the optimality of S', the cost of S' is not greater than the cost of OPT$'$, and therefore it is at most opt$/p$.

Consider an optimal solution OPT for the instance of AR. We construct a (fractional) flow in G' of capacity $|Q|$ corresponding to OPT in the following way. We send a flow of 1 along each edge $\{v_s, q_{i,j}\}$ leaving the source. At each vertex $q_{i,j} \in Q$, this flow of 1 is split into $k \cdot p$ equal parts, one for each vertex; recall that each $Q^\star_{i,j}$ has exactly $k \cdot p$ vertices. For each vertex $u \in Q^\star_{i,j}$, we send the amount of $1/(k \cdot p)$ flow along the edge $\{q_{i,j}, l_v\}$, where v is the airport serving u in the solution OPT. Finally, for every vertex $l_v \in L$ we forward all the received flow (which by feasibility of OPT cannot be greater than $\frac{1}{k \cdot p} \cdot k = 1/p$) along the outgoing edge $\{l_v, v_t\}$ to the sink v_t.

The constructed flow OPT$'$ has capacity $|Q|$ and is feasible, as the only edges where the amount of flow might be greater than 1 are the edges of E_L, and in the reasoning above we have shown an upper bound of $1/p$ on the flow on these edges.

We will now upper bound the cost of OPT$'$ with respect to the cost of OPT. Let opt $= \texttt{cost}_e(\text{OPT}) + \texttt{cost}_a(\text{OPT})$, where $\texttt{cost}_e(\text{OPT})$ is the edge cost of OPT and $\texttt{cost}_a(\text{OPT})$ is the airport cost of OPT. For any $v \in V(G)$, let $b(v)$ denote the airport serving v in OPT. As each airport serves at most k cities, we have $\texttt{cost}_e(\text{OPT}) \geq \frac{1}{k} \sum_{v \in V(G)} \text{len}(v, b(v))$.

For any $l_v \in L$, OPT$'$ sends some flow along the edge $\{l_v, v_t\}$ only when OPT opens an airport at v. As the capacity of each edge of E_L in G' is $1/p$, the cost of OPT$'$ on the edges of E_L is therefore at most $\texttt{cost}_a(\text{OPT})/p$. We will now upper bound the cost of OPT$'$ on the edges of E_{QL}. By the construction of OPT$'$, this cost equals

$$\sum_{q_{i,j} \in Q} \sum_{u \in Q^\star_{i,j}} \frac{d(q_{i,j}, b(u))}{k \cdot p} \leq \sum_{q_{i,j} \in Q} \sum_{u \in Q^\star_{i,j}} \frac{\text{len}(u, b(u))}{k \cdot p} \leq \sum_{u \in V(G)} \frac{\text{len}(u, b(u))}{k \cdot p} \leq \frac{\texttt{cost}_e(\text{OPT})}{p}.$$

The edges of G' which are in E_Q have cost 0, so they do not contribute towards the cost. Therefore $\texttt{cost}(\text{OPT}') \leq \text{opt}/p$. As S' is an optimal solution for the min-cost max-flow instance, we get $\texttt{cost}(S') \leq \texttt{cost}(\text{OPT}') \leq \text{opt}/p$. ◀

From the integral min-cost max-flow solution S' we construct S_2 as follows. We start by taking $S_2 = S_1$. Then, we open the airports $u \in V(G)$ which have not been opened by S_1, and for which S' has flow at least 1 on the corresponding edge $\{l_u, v_t\}$. Then, for each

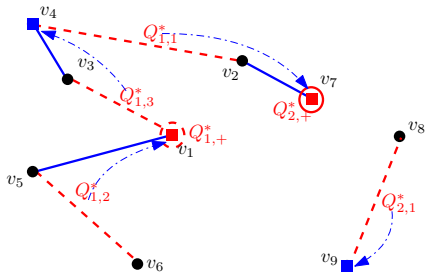


Figure 4 An infeasible solution S_2 for the instance I of AR from Figure 1 is pictured in red (dashed) and blue (solid). The new airports (blue squares) have been opened at vertices v_4 and v_9. The blue edges connect the subpaths $Q^\star_{i,j}$ with the assigned airports (the assignment is pictured by the dashed arrows). The solution is infeasible, as the airports serving $Q^\star_{1,2}$ and $Q^\star_{1,3}$ are in different components than the corresponding subpaths. (In the drawing, the components corresponding to the airports v_1, v_4 and v_7 got connected.) Note also, that the airport v_7 now serves three cities (v_2, v_4 and v_7), and therefore requires resource augmentation.

subpath $Q^\star_{i,j}$ we find the unique vertex $u \in V(G)$, such that S' uses the edge $\{q_{i,j}, l_u\}$. Note that in this case S_2 must have an airport at u. Let $v_{i,j}$ be the vertex of $Q^\star_{i,j}$ minimizing the distance $\mathrm{len}(v_{i,j}, u)$. We add to S_2 the edge $\{v_{i,j}, u\}$, and we assign $Q^\star_{i,j}$ to the airport u (i.e., the cities from $Q^\star_{i,j}$ will be served by u). See Figure 4 for an example.

Note that in this construction each subpath $Q^\star_{i,j}$ is assigned to an airport of S_2, and therefore all cities from $Q^\star_{i,j}$ are served. As the capacity of the edges $\{l_u, v_t\}$ is $1/p$, at most $1/p$ subpaths get connected to one airport. We can show the following.

▶ **Lemma 7.** *The solution S_2 has the following properties: (i) $\mathtt{cost}(S_2) \le (2 + 1/p)opt$, and (ii) each city is served by some airport, and each airport serves strictly less than $(1 + p) \cdot k$ many cities.*

▶ **Observation 3.** *The solution S_2 is still not feasible. For any vertex $u \in V(G)$ it may happen that the city u is served by an airport at some vertex $v \in V(G)$ with $v \neq u$, and at the same time the airport at u has been opened and serves some other component. In particular, a single component might contain a large number of airports, each of them serving a different component. (Then, when considering the edges of S_2, such components create a single connected component of S_2.) This is not consistent with the definition of the* AR *problem, where the airport serving a component must belong to this component.*

Step 3: Making the solution feasible. Solution S_3. In this final step we show how we can transform the solution S_2 to a feasible solution S_3, with only a small increase in cost and while increasing the size of each component by at most one vertex.

We consider the components of S_2 one by one. For each component T_ℓ, we consider the cities which belong to T_ℓ, as well as the airport $u \in V(G)$ serving the cities from T_ℓ. If the city u belongs to T_ℓ (i.e., it is served by the airport at u), we do not make any changes. Otherwise, we re-assign the city u to T_ℓ. We do that by removing u from its current component, and by adding it to T_ℓ. We denote the resulting solution by S_3.

▶ **Lemma 8.** *The re-assignment can be performed so that the solution S_3 has the following properties. (i) Each airport in S_3 serves at most $(1+p) \cdot k$ cities, (ii) $\mathtt{cost}(S_3) \le \frac{4}{3}\mathtt{cost}(S_2)$, and (iii) S_3 is a feasible solution for AR_{1+p}.*

Theorem 1 follows from Lemmas 7 and 8.

3 The Euclidean Case

In this section we focus on AR in the two-dimensional Euclidean space. We first show in Section 3.1 that if we allow a small resource augmentation of the airport capacities, we are able to obtain a quasi-polynomial-time approximation scheme (QPTAS). We then, in Section 3.2, present a polynomial time $(1+\epsilon)$-approximation algorithm for a relaxed version of the problem that allows components to have size larger than k, but where each component must have enough airports in order to serve all clients. This approximation algorithm is then used in Section 3.3 in order to give a constant factor approximation algorithm for the general AR in two-dimensional Euclidean space, which is our main result for the section.

3.1 A QPTAS with a Small Resource Augmentation

In this section we give a sketch of the proof of Theorem 3, i.e., a QPTAS for two-dimensional Euclidean $\text{AR}_{(1+\mu)}$ for any fixed $\mu > 0$. Our algorithm is based on Arora's scheme [5].

First, using standard techniques, we partition the problem instance into independent subinstances and perform perturbation. This step reduces the original problem instance into a collection of independent subinstances, where each instance has all input points at points with integer coordinates, allows Steiner vertices only at points with integer coordinates, and is bounded by a polynomially sized bounding box. That increases the cost of the obtained solution only by a negligible factor.

Next, as in Arora's scheme [5], we introduce a shifted quadtree, which recursively decomposes the input box into smaller and smaller subsquares (called dissection squares), ending with leaf squares which contain only one point with integer coordinates each. Then, at the boundary of each dissection square we introduce a logarithmic number of equidistant portals. We then show that, by increasing the cost of the obtained solution only by a negligible factor, we can consider solutions where edges cross the boundary of the dissection squares only at the portals. By losing another negligible factor, we further restrict the solutions so that every component is $O(1)$-light, i.e., it crosses the boundary of each dissection square at at most $O(1)$ portals.

For each dissection square C, with each component T of the solution, we associate the *type* of T, which specifies the $O(1)$ portals used by T, a partition of these portals into sets such that each set corresponds to a connected component of $T \cap C$, information whether there is an open airport in T within C, and the number of points from $V(G)$ in $T \cap C$ rounded down to the nearest *threshold*, where the number of thresholds is polylogarithmic. That gives a polylogarithmic number of types of components.

This allows us to use a dynamic program that finds a near-optimal solution for $\text{AR}_{(1+\mu)}$ for any constant $\mu > 0$. In the dynamic program, we have a set of possible *configurations* for each dissection cell C, where each configuration specifies the number of components of each of the polylogarithmic number of types. Therefore, the number of configurations is quasi-polynomial. For leaf dissection squares, we can find an optimal solution satisfying each configuration. Then, the DP proceeds bottom-up, computing solutions for all the configurations for larger dissection squares, based on the solutions for the subsquares. We can show that, by choosing the number of thresholds appropriately, the resource augmentation required for the DP solution can be upper-bounded by $1 + \mu$.

3.2 Relaxed AR: Allowing components with multiple airports

In this section we define a relaxed version of AR, which we denote by AR′. The difference between AR and AR′ is that each component of AR′ can contain multiple airports and multiple copies of the same edges. Moreover, each component allows routing all customers to the airports, where each airport serves at most k customers, and each copy of an edge can be used by at most k customers. As in the case for AR, AR′ can also use Steiner vertices.

Intuitively, AR′ is a relaxation of AR where cities assigned to different airports can share the same edges. We now define the problem formally.

▶ **Definition 9.** In the problem AR′ we are given a set of points $V(G)$ on the Euclidean plane, together with a cost $a(v)$ for each point $v \in V(G)$, and an integral capacity parameter k. A feasible solution is a subset of vertices $A \subseteq V(G)$ and a network consisting of edges $E(G)$ that allows routing the flow of one unit from each point in $V(G)$ to the points in A, such that (i) each edge in the network has capacity k, and (ii) each point from A can receive at most k units of flow. The network can use each point on the Euclidean plane as Steiner vertices, and parallel edges are allowed.

The goal is to find a feasible solution minimizing the total cost of the network, i.e., the value of $\sum_{v\in A} a(v) + \sum_{e\in E(G)} \mathrm{len}(E(G))$.

We obtain a polynomial time algorithm that for every input instance I of AR finds a solution to instance I of AR′ with cost of at most $(1+\epsilon)\mathrm{opt}_{\mathrm{AR}}$, where $\mathrm{opt}_{\mathrm{AR}}$ is the cost of an optimal solution to I for AR. The algorithm is based on a dynamic programming formulation, and builds on Arora's PTAS for the Euclidean TSP [5].

With each solution for an instance I of AR′ we can associate a network flow f that defines an assignment of cities to the airports within each component of the solution. Such flow can be computed efficiently.

3.3 A Constant-Factor Approximation Algorithm

In this section we use the algorithm from Section 3.2 as a building-block of a constant-factor approximation algorithm for the original AR problem in the two-dimensional Euclidean space. Note that this algorithm does not require resource augmentation.

We proceed similarly as in Steps 1 and 2 from Section 2, cutting the initial solution into pieces, and matching the pieces to the airports.

Step 0: Obtaining solution S_0. Given an instance I of AR, we run the algorithm from Section 3.2 on I, obtaining a solution S_0. Although S_0 is not feasible in general, as it may contain components with more than k cities and more than one airport, it is a good starting point, as we can upper bound its cost by opt.

▶ **Lemma 10.** *Consider the solution S_0 for an instance I of two-dimensional Euclidean* AR*. Let $\{C_1, C_2, \ldots C_z\}$ be the set of connected components of S_0, where each component C_i contains h_i airports. The following holds: (i) $\mathtt{cost}(S_0) \leq (1+\epsilon)\mathrm{opt}$, and (ii) for each component C_i, the number of points of C_i which are in $V(G)$ satisfies: $|C_i| \leq k \cdot h_i$.*

We now show how to transform S_0 into a feasible solution for AR. For each connected component C_i of S_0 with $h_i > 1$, we proceed in two further steps that slightly resemble steps from Section 2. However, we have to be more careful in order to avoid resource augmentation. In the first step we will "cut" each such component C_i into singleton components containing the airports of S_0, and paths containing at most $k-1$ cities (with no airports) each. In the

second step, then we will develop an algorithm that matches these paths to airports without increasing the cost by more than a constant factor.

Before we start, we perform the following operations. First, we compute a flow f in S_0, and we modify f into a flow f', so that each airport v_i serves the city at v_i. We will modify the instance I_0 into an instance I'_0, and the solution S_0 into S'_0, so that each airport serves exactly k cities. We do that by introducing for each airport v_i *additional cities*, coincident with the airport v_i and being served by v_i, so that v_i serves exactly k cities. The cost of S'_0 is then the same as the cost of S_0 (as the additional cities are served for free). We will transform S'_0 into a feasible solution for the instance I' of AR, while increasing its cost only by a constant factor, and then by dropping the additional cities we will obtain a solution for the instance I.

Step 1: Cutting each component C_i. Solution S_1. Consider a connected component C_i of S'_0 (and therefore also of S_0), which contains $|C_i|$ cities. We first transform C_i into a path P_i that visits all vertices of C_i that do not contain an open airport. We do this by first doubling all edges of C_i, obtaining an Eulerian tour on it, shortcutting the resulting tour so that it only visits cities that do not have an airport[4], and then removing a single edge from the tour. We have

$$\sum_{i=1}^{z} \texttt{cost}(P_i) \leq 2 \cdot \texttt{cost}_e(S_0), \tag{1}$$

where $\texttt{cost}_e(S_0)$ refers to the edge cost of the solution S_0.

We now break each path P_i into a collection of h_i subpaths $\{Q_{i,1}, Q_{i,2}, \dots Q_{i,h_i}\}$, such that each subpath $Q_{i,j}$ contains exactly $k-1$ cities of I'. Note that we can do this, as in S'_0 the component C_i contains exactly $k \cdot h_i$ cities (including the additional cities), and after removing the airports the path P_i contains $(k-1) \cdot h_i$ cities.

Let S_1 be a solution consisting of the airports open by S_0 (and therefore also by S'_0) that form a singleton components $\{v_i\}$ and serve the cities at v_i, and of the paths $Q_{i,j}$ that do not contain open airports and contain exactly $k-1$ cities of I' each.[5] We now upper bound the cost of S_1.

▶ **Lemma 11.** $\texttt{cost}(S_1) \leq 2 \cdot \texttt{cost}(S_0)$.

Step 2: Matching the subpaths $Q_{i,j}$ to the airports within each component C_i. Solution S_2. In order to assign the subpaths $Q_{i,j}$ to the airports of component C_i, we develop an instance of of minimum cost perfect matching in a bipartite graph. This can be seen as a simplified version of our network flow construction in Section 2. We do this for each component C_i separately.

Consider a component C_i of S'_0 (and therefore also of S_0). We construct a bipartite graph G'_i as follows. For each subpath $Q_{i,j}$ we introduce a vertex q_j, and denote the set of such vertices by Q. For each vertex $u \in C_i$ with an airport in S_0, introduce a vertex l_u and denote this set of such vertices by L. Now we form a complete bipartite graph G'_i, where the vertices of Q are at one side of the bipartition, and the vertices of L are at the other side. An edge $\{q_j, l_u\}$ has cost equal to the minimum distance between the subpath $q_{i,j}$ and the vertex u. More formally, the cost of the edge $\{q_j, l_u\}$ equals $\min_{v \in Q_{i,j}} \text{len}(\{v, u\})$.

[4] Note that if there are additional cities coincident with a city v_i with an airport, the path should not visit the city v_i, but it should still visit the coincident additional cities.

[5] Note that the paths $Q_{i,j}$ can have coincident endpoints, if they visit the coincident additional cities.

We construct the solution S_2 as follows. We start by setting $S_2 = S_1$. For each component C_i of S_0 we compute (in polynomial time) a min-cost perfect matching for the graph G'_i described above. Such a matching exists, as C_i has h_i airports and h_i subpaths, therefore both parts of the bipartition have equal size. For each matching edge $\{q_j, l_u\} \in Q \times L$ we add to the solution S_2 an edge $\{v, u\}$, where $v \in Q_{i,j}$ has minimum distance to u out of all vertices of $Q_{i,j}$. We then remove the additional cities from each component of S_2.

▶ **Lemma 12.** *Solution S_2 has the following properties: (i) it is a feasible solution for the* AR *instance (without resource augmentation), and (ii)* $\texttt{cost}(S_2) \leq \left(2 + \frac{k}{k-1}\right)(1+\epsilon)opt$.

The proof of Theorem 2 now directly follows from Lemma 12 and by substituting ϵ with $\epsilon/4$.

References

1. Anna Adamaszek, Antonios Antoniadis, and Tobias Mömke. Airports and railways: Facility location meets network design. In *STACS*, volume 47 of *LIPIcs*, pages 6:1–6:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
2. Anna Adamaszek, Artur Czumaj, Andrzej Lingas, and Jakub Onufry Wojtaszczyk. Approximation schemes for capacitated geometric network design. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 2011. doi:10.1007/978-3-642-22006-7_3.
3. Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.
4. Hyung-Chan An, Mohit Singh, and Ola Svensson. LP-based algorithms for capacitated facility location. In *FOCS*, pages 256–265. IEEE Computer Society, 2014.
5. Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.
6. Babak Behsaz, Mohammad R. Salavatipour, and Zoya Svitkina. New approximation algorithms for the unsplittable capacitated facility location problem. *Algorithmica*, 75(1):53–83, 2016.
7. Raja Jothi and Balaji Raghavachari. Approximation algorithms for the capacitated minimum spanning tree problem and its variants in network design. *ACM Transactions on Algorithms (TALG)*, 1(2):265–282, 2005.
8. Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Inf. Comput.*, 222:45–58, 2013.
9. Jens Maßberg and Jens Vygen. Approximation algorithms for a facility location problem with service capacities. *ACM Trans. Algorithms*, 4(4):50:1–50:15, 2008.
10. R. Ravi and Amitabh Sinha. Approximation algorithms for problems combining facility location and network design. *Operations Research*, 54(1):73–81, 2006. doi:10.1287/opre.1050.0228.
11. Alexander Schrijver. *Combinatorial Optimization*. Springer, 2003.

Property Testing for Bounded Degree Databases

Isolde Adler
University of Leeds, School of Computing, Leeds, UK
i.m.adler@leeds.ac.uk

Frederik Harwath
Institut für Informatik, Goethe-Universität, 60054 Frankfurt, Germany
harwath@cs.uni-frankfurt.de

Abstract

Aiming at extremely efficient algorithms for big data sets, we introduce property testing of relational databases of bounded degree. Our model generalises the bounded degree model for graphs (Goldreich and Ron, STOC 1997). We prove that in this model, if the databases have bounded tree-width, then every query definable in monadic second-order logic with modulo counting is testable with a constant number of oracle queries and polylogarithmic running time. This is the first logical meta-theorem in property testing of sparse models. Furthermore, we discuss conditions for the existence of uniform and non-uniform testers.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms, Theory of computation → Logic and databases, Mathematics of computing → Graph algorithms

Keywords and phrases Logic and Databases, Property Testing, Logical Meta-theorems, Bounded Degree Model, Sublinear Algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.6

Acknowledgements We would like to thank Tomáš Gavenčiak for critical discussions.

1 Introduction

As technology advances, there is an increased need for new *extremely efficient* algorithms that deal with typical computational problems on ever larger databases. Approximate Query Processing [4] addresses this by seeking to provide approximate answers to queries at a fraction of the cost of traditional query execution, thus opening the path for revealing new insights into voluminous data sets. In this spirit, we study a relaxation of Boolean Query Evaluation on relational databases of bounded degree, including performance and (probabilistic) accuracy guarantees.

The problem of Boolean Query Evaluation asks, for a Boolean query Q and a relational database $\mathcal{D}$, whether $\mathcal{D}$ satisfies Q. Relaxing this problem, we want to distinguish with high probability correctly the case that $\mathcal{D}$ satisfies Q from the case that $\mathcal{D}$ is ϵ-*far* from satisfying Q, i. e. from the case that we need to modify (add / delete) more than an ϵ-fraction of the tuples in relations of $\mathcal{D}$ to make the database satisfy Q. For $\epsilon \in (0, 1]$, an ϵ-*tester* is a probabilistic algorithm that makes this distinction by only looking at a small number of small parts of the input database. More precisely, the ϵ-tester receives the size n of the input database and has oracle access to the database. For each given element of the domain, the tester can query the oracle for tuples (in any of the relations) containing the element. The *query complexity* $q(n)$ of the ϵ-tester is the maximum number of oracle queries performed, over all input databases on n elements.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 6; pp. 6:1–6:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


In this paper we only consider databases of degree bounded by a constant d, i. e. every element participates in at most d tuples in relations, and we assume that the oracle can answer queries in constant time. Due to the degree bound, a database $\mathcal{D}$ on n elements has at most dn tuples in relations. We say $\mathcal{D}$ is ϵ-*far* from satisfying query Q, if we need to delete or insert more than ϵdn tuples in $\mathcal{D}$ to satisfy Q. Given our interest in highly efficient algorithms, we study testers with *constant* query complexity q, i. e. q is independent of n (but may depend on ϵ and the degree d). Here, we view a *Boolean query* as an isomorphism closed class of relational databases. We use the term *property* instead of Boolean query. We say that a property Q is *testable*, if for every ϵ, there is an ϵ-tester for Q with constant (oracle) query complexity.

In [2], Alon, Krivelevich, Newman and Szegedy proposed a systematic study of the testability of logically defined properties. We obtain the first such *logical meta-theorem* for property testing in the bounded degree model, which relates logical definability to time efficient uniform testability. This is the main result of our paper. It can be thought of as testability-analogue of the well-known theorem Courcelle [6], which states that each property of relational databases which is definable in *monadic second-order logic with counting* (CMSO) is decidable in linear time on relational databases of bounded tree-width. We show that each such property is testable on structures of bounded degree and bounded tree-width.

Our model extends the bounded-degree model for property-testing of graphs, introduced by Goldreich and Ron [15], and the bidirectional model for directed graphs of [3, 9]. In [21], Newman and Sohler showed that every *hyperfinite* graph property is testable. This includes properties such as planarity and minor-closed graph classes. Their testers are *non-uniform* in the number n of vertices of the input graph. (Indeed, it is not hard to come up with a property of (edgeless) graphs that is undecidable but hyperfinite, and hence testable.) We generalise this result to databases of bounded degree over a fixed finite signature. We then study conditions that allow for *time efficient uniform* testers, and we characterise both non-uniform testability and uniform testability. We introduce a combinatorial criterion which characterises non-uniform testability with constant query complexity. This criterion which we call *locality* is similar to the concept of *Hanf locality* from mathematical logic which provides a combinatorial method for proving non-definability results for first-order logic and its counting extensions (cf. [17]). It captures the intuition that testability of a property with constant query complexity means that the property is determined by the distribution of substructures of a constant radius r (r-discs) in a structure. We also introduce a concept which we call *effective locality* and which characterises uniform testability. This pinpoints that uniform testability of a property is closely related to solving a relaxation of a *realisability problem* of r-disc distributions. This problem asks for a given distribution D and a natural number n whether there exists a structure on n elements satisfying the property and where the r-discs are distributed according to D.

Techniques. Similar to the evaluation of scale independent queries in [12], our algorithms only explore a constant size sub-database before outputting the answer. For this, we explicitly require the database to have bounded degree, whereas scale-independent queries use a degree restriction implicit in the access schema.

The proof of the CMSO-result involves a number of steps. By our characterisation of uniform testability it suffices to show that all CMSO-definable properties are effectively local. To prove this, we generalise the Local-Global Theorem [21, Thm. 3.1] from graphs to databases. We then show that the realisability problem can be solved efficiently. For this, we use the fact that many-sorted spectra of CMSO formulas on bounded tree-width are

semilinear, a result by Fischer and Makowsky [13], that is based on Parikh's Theorem [22]. This allows to encode the problem as an integer linear program (ILP) with a fixed number of variables and inequalities. Using a result of Lenstra [18] for solving such ILPs we obtain polylogarithmic running time.

Let us remark that some work goes into generalising tools from property testing on graphs to relational databases. For this we could reprove the relevant results in our slightly more general setting (this is not done in this extended abstract for space constraints). A different approach would be to encode relational structures as (coloured) incidence graphs, and apply the known results for graphs. This can be done, because bounded degree, bounded tree-width and hyperfiniteness carry over to incidence graphs. Nevertheless, it involves some technicalities, in particular when simulating testing algorithms on incidence graphs. We plan to take this approach in the journal version.

Structure of the paper. Section 2 introduces the notation and general definitions used throughout the paper. Section 3 is devoted to the discussion of our model, including illustrating examples. Section 4 contains our logical meta-theorem, together with the generalisations of known results about graphs to databases. Section 5 characterises testability by our notion of locality, and uniform testability by *effective locality*. We conclude in Section 6.

Proofs. Several proofs are omitted due to space contraints for this extended abstract. We indicate this by $(*)$.

2 Preliminaries

We let $\mathbb{N}$ denote the set of natural numbers including 0. For each $n \in \mathbb{N}$ with $n \geq 1$, we let $[n] := [1, n] \cap \mathbb{N}$. A *partition* of a set A is a collection P of non-empty, pairwise disjoint subsets of A whose union is A. For an element $a \in A$ we let $P[a]$ denote the unique member of P containing a.

Relational structures. In this paper, we consider relational databases as *finite relational structures* over *finite signatures*. A *signature* is a finite set $\sigma := \{R_1, \dots, R_\ell\}$ of *relation symbols*, each of which has an *arity*, $\mathrm{ar}(R_i) \in \mathbb{N}$. We let $\mathrm{ar}(\sigma)$ denote the maximum arity of the relation symbols contained in σ. A *σ-structure* is a tuple $\mathcal{A} := (A, R_1^{\mathcal{A}}, \dots, R_\ell^{\mathcal{A}})$ where A is a finite set, called the *universe* of $\mathcal{A}$, and $R_i^{\mathcal{A}}$ is a $\mathrm{ar}(R_i)$-ary relation on A. The members of the universe are the *elements* of $\mathcal{A}$. We let $|\mathcal{A}| := n$ denote the number n of elements of $\mathcal{A}$. We assume that all structures are linearly ordered in some way or, equivalently, that the universe of a structure with n elements is $[n]$. We extend the linear order on $\mathcal{A}$ to a linear order on each relation of $\mathcal{A}$ via lexicographic ordering. We say that a σ-structure $\mathcal{B}$ is a *substructure* of another σ-structure $\mathcal{A}$, if $\mathcal{B}$ can be obtained from $\mathcal{A}$ by deleting a (possibly empty) set of elements from $\mathcal{A}$ and a (possibly empty) set of tuples from the relations of $\mathcal{A}$. Note that we have to relabel the elements of $\mathcal{B}$ after deleting elements to be sure that the universe of $\mathcal{B}$ is still an initial segment of the natural numbers. A substructure of $\mathcal{A}$ is *induced* by a set $M \subseteq A$ if it can be obtained from $\mathcal{A}$ by deleting all elements of $A \setminus M$ (and all incident tuples). We denote such a structure by $\mathcal{A}[M]$. A *property* is a class of structures that is closed under isomorphism. Indeed, we only consider isomorphism closed classes of structures. For each class C of structures, we define $\mathrm{C}|n := \{\mathcal{A} \in \mathrm{C} : |A| = n\}$. The *degree* $\deg(a)$ of an element a in a structure $\mathcal{A}$ is the total number of tuples in all relations which

contain a. We define the *degree* $\deg(\mathcal{A})$ of a structure $\mathcal{A}$ as the maximum degree of its elements. A class C of structures has *bounded degree* d if $\deg(\mathcal{A}) \leq d$ for each $\mathcal{A} \in \mathrm{C}$. A class C is of structures is *closed under removing tuples*, if for every structure $\mathcal{A} \in \mathrm{C}$, the structure $\mathcal{A}'$ obtained from $\mathcal{A}$ by deleting a tuple from some relation of $\mathcal{A}$ is also a member of C.

A class C has *bounded degree* if there exists a number d such that C has bounded degree d. We write $\mathrm{C}_{\sigma,d}$ for the class of all σ-structures of degree at most d. Most of the time, σ will be fixed and we omit it from this notation. The *Gaifman graph* of a structure $\mathcal{A}$ is the undirected graph $\mathcal{G}(\mathcal{A}) = (A, E)$, where $\{x, y\} \in E$ if x and y occur together in a tuple belonging to some relation of $\mathcal{A}$. Observe that our notion of bounded degree classes coincides with the notion where the degree of $\mathcal{A}$ is defined as the (usual graph-theoretical) maximum degree of $\mathcal{G}(\mathcal{A})$.

We transfer the usual graph theoretic (shortest path) distance and related notions (e.g. radius, connectivity) from Gaifman graphs $\mathcal{G}(\mathcal{A})$ to structures $\mathcal{A}$. A non-empty induced substructure $\mathcal{B}$ of $\mathcal{A}$ is a *connected component* of $\mathcal{A}$, if $\mathcal{G}(\mathcal{B})$ is a connected component of $\mathcal{G}(\mathcal{A})$. For each $r \in \mathbb{N}$, an *r-disc* is a pair $(\mathcal{A}, a)$ where $\mathcal{A}$ is a relational structure of radius at most r and a is a central element of $\mathcal{A}$. Two r-discs $(\mathcal{A}, a)$ and $(\mathcal{B}, b)$ are *isomorphic* (written $(\mathcal{A}, a) \cong (\mathcal{B}, b)$) if there is an isomorphism of $\mathcal{A}$ and $\mathcal{B}$ which maps a to b. An *r-type* is an $\cong$-equivalence-class of r-discs. The number of *r-types* is bounded by a constant which depends only on the signature σ and the fixed degree bound d. We write $c(r)$ to denote this constant. For a given structure $\mathcal{A}$ and an element a of $\mathcal{A}$, the *r-neighbourhood of a in $\mathcal{A}$*, written $N_r^{\mathcal{A}}(a)$ is the set of all elements with distance at most r to a. The *r-disc around a in $\mathcal{A}$* is the structure $\mathcal{D}_r^{\mathcal{A}}(a) := (\mathcal{A}[N_r^{\mathcal{A}}(a)], a)$. We say that an element a *realises* an r-disc-type τ, if $\mathcal{D}_r^{\mathcal{A}}(a) \in \tau$. For each structure $\mathcal{A}$, the *r-histogram of $\mathcal{A}$* denotes the vector $\mathrm{h}_r(\mathcal{A})$ with $c(r)$ components, indexed by the r-disc types, where the component corresponding to type τ contains the number of elements of $\mathcal{A}$ which realise τ. More generally, we call a vector $\bar{v}$ an *r-histogram with n elements* if it has $c(r)$ components and if $n = \sum_{i \leq c(r)} \bar{v}[i]$. For every class of structures C and each $r \in \mathbb{N}$, we let $\mathrm{H}_r(\mathrm{C}) := \{\mathrm{h}_r(\mathcal{A}) : \mathcal{A} \in \mathrm{C}\}$. If there exists a structure $\mathcal{A} \in \mathrm{C}$ such that $\bar{v} = \mathrm{h}_r(\mathcal{A})$, we say that $\bar{v}$ is C*-realisable*. For each r-histogram $\bar{v}$ and each r-disc $\mathcal{D}$, we let $\bar{v}[\mathcal{D}] := \bar{v}[i]$ where τ_i is the unique r-disc type with $\mathcal{D} \in \tau_i$. For an r-histogram $\bar{v}$ with n elements, the vector $\bar{v}/n$ specifies a *distribution* of r-types.

3 The Model

Algorithms with direct access. Let σ be a signature. We consider algorithms which process σ-structures of bounded degree d. An algorithm that processes $\mathcal{A}$ does not obtain an encoding of $\mathcal{A}$ as a bit string in the usual way. Instead, it has direct access to $\mathcal{A}$ using an *oracle* which answers queries about the relations of $\mathcal{A}$ in constant time. The (usual) input of the algorithm consists of auxiliary information. In our case, this will always be the size n of the universe of $\mathcal{A}$ in binary representation. The oracle accepts queries of the form (R, i, j), for $R \in \sigma$, $i \leq n$, and $j \leq \deg(\mathcal{A})$, to which it responds with the j-th tuple of the relation $R^{\mathcal{A}}$ which contains the i-th element, or with $\perp$ if there are strictly less than j tuples in $R^{\mathcal{A}}$ which contain i. The *running time* of the algorithm is defined as usual, i.e. with respect to the auxiliary input n. As common in property testing, we use a uniform cost model, i. e. we assume that all basic arithmetic operations including random sampling can be performed in constant time, regardless of the size of the numbers involved.

Distance. For two graphs G and H, both with n vertices, $\mathrm{dist}(G, H)$ denotes the minimum number of edges that have to be inserted or removed from G and H to make G and H isomorphic. For two σ-structures $\mathcal{A}, \mathcal{B}$ with the same number n of elements, $\mathrm{dist}(\mathcal{A}, \mathcal{B})$

denotes the minimum number of tuples that have to be inserted or removed from $\mathcal{A}$ and $\mathcal{B}$ to make $\mathcal{A}$ and $\mathcal{B}$ isomorphic. Let $\epsilon \in [0,1]$ and $d \in \mathbb{N}$. If $\deg(\mathcal{A}), \deg(\mathcal{B}) \leq d$, we say that $\mathcal{A}$ and $\mathcal{B}$ are ϵ-*close* (with respect to d) if $\operatorname{dist}(\mathcal{A},\mathcal{B}) \leq \epsilon dn$. Usually, the number d is fixed and will not be mentioned. If $\mathcal{A},\mathcal{B}$ are not ϵ-*close*, then they are ϵ-*far*. We say that a structure $\mathcal{A}$ is ϵ-*close* to a property P if $\mathcal{A}$ is ϵ-close to some $\mathcal{B} \in \mathrm{P}$. Otherwise, $\mathcal{A}$ is ϵ-*far* from P. Given a class of structures C, we write ϵ-close$_\mathrm{C}$(P) and ϵ-far$_\mathrm{C}$(P) for the set of σ-structures from C which are ϵ-close and ϵ-far to P, respectively. Usually, we omit C from this notation if it can be inferred from the context. Note that since P is closed under isomorphism, both ϵ-close(P) and ϵ-far(P) are closed under isomorphism as well.

Degree bounds and distances carry over from structures to their Gaifman graphs as follows.

▶ **Lemma 1** (∗). *Let σ be a signature of maximum arity $\alpha := \operatorname{ar}(\sigma)$ and let $\mathcal{A},\mathcal{B}$ be σ-structures of degree at most $d \geq 1$. Then $\deg(\mathcal{G}(\mathcal{A})) \leq d\alpha$, and $\operatorname{dist}(\mathcal{G}(\mathcal{A}),\mathcal{G}(\mathcal{B})) \leq \binom{\alpha}{2} \operatorname{dist}(\mathcal{A},\mathcal{B})$.*

▶ **Definition 2** (ϵ-tester)**.** Let σ be a signature and let $d \in \mathbb{N}$. Let C be a class of σ-structures of bounded degree d and let $\mathrm{P} \subseteq \mathrm{C}$ be a property. An ϵ-*tester for* P *on* C is a probabilistic algorithm with direct access to σ-structures. Given oracle access to a σ-structure $\mathcal{A} \in \mathrm{C}$ and given $n := |A|$ as auxiliary input, the algorithm does the following:

1. If $\mathcal{A} \in \mathrm{P}$, then the tester accepts with probability at least $\frac{2}{3}$.
2. If $\mathcal{A} \in \epsilon$-far(P), then the tester rejects with probability at least $\frac{2}{3}$.

The *query complexity* of a tester is the maximum number of oracle queries made. A property $\mathrm{P} \subseteq \mathrm{C}$ is *uniformly testable in time $f(n)$ on* C, if for each $\epsilon \in (0,1]$ there is an ϵ-tester for P which has *constant query-complexity* (i. e. independent of n) and whose running time on structures with n elements is $f(n)$. Note that this tester must work for all n.

A property P is *non-uniformly testable*, if for each n, the class $\mathrm{P}|n$ is testable on $\mathrm{C}|n$, i.e. there may be a different tester for each input size. (Note that this definition of uniform testability is non-standard. Some authors define uniform testability as being uniform with respect to ϵ.)

Our definitions subsume the bounded degree model of graph property testing. Let $\sigma := \{E\}$, where E is a binary relation symbol, and let C denote the class of all undirected graphs (viewed as symmetric, irreflexive directed graphs) of degree at most d. Then, up to a constant factor in the query- and time complexity, all results about testability in the bounded degree graph model carry over to testability in our model. In the same way, our model generalises the bidirectional model for directed graphs of bounded degree introduced in [3]. We illustrate the model by two simple examples.

▶ **Example 3** (Keys)**.** A component of a relation is a *key* if there are no two tuples in the relation which contain the same value in this component. Let $\sigma := \{R\}$ be a signature. Consider the property Key containing all σ-structures $\mathcal{A}$ where the first component of $R^\mathcal{A}$ is a key. Let $d \in \mathbb{N}$. We show that on the class $\mathrm{C}_{\sigma,d}$, Key is uniformly testable with constant running time. Let $\epsilon \in (0,1]$. Given oracle access to a σ-structure $\mathcal{A}$ on n elements, the ϵ-tester proceeds as follows.

1. Sample $\alpha := \log_{1-\epsilon} \frac{1}{3}$ elements from $[n]$ uniformly and independently.
2. For each of these elements i, perform the queries $(R,i,1),\ldots,(R,i,d)$ to obtain all tuples of $R^\mathcal{A}$ which contain i.
3. If, in the previous step, two tuples with the same elements in their first components are found, the tester rejects $\mathcal{A}$. Otherwise, $\mathcal{A}$ is accepted.

Call a tuple belonging to $R^{\mathcal{A}}$ *bad* if $R^{\mathcal{A}}$ contains another tuple with the same first component. An element is *bad* if it occurs in the first component of a bad tuple. The tester clearly accepts $\mathcal{A}$ if $\mathcal{A} \in$ KEY, i.e. it contains no bad tuples. Suppose that $\mathcal{A}$ is ϵ-far from KEY. Then there are at least ϵdn bad tuples. Since $\mathcal{A}$ has degree at most d, at least ϵn different elements occur in the first components of these bad tuples. The probability that a uniformly random element of $\mathcal{A}$ is bad is hence at least ϵ. The probability that our sample of α independently selected elements contains no bad element is $(1-\epsilon)^{\alpha} \leq \frac{1}{3}$. Hence, with probability at least $\frac{2}{3}$ the algorithm samples a bad element i. Since i is bad, there are at least two tuples which contain i and hence the algorithm rejects $\mathcal{A}$. The running time is constant.

▶ **Example 4** (Symmetry)**.** A k-ary relation is *symmetric* if for each tuple $\bar{a} := (a_1, \ldots, a_k)$ of the relation and each permutation π of $[k]$, the tuple $\pi(\bar{a}) := (a_{\pi(1)}, \ldots, a_{\pi(k)})$ is also contained in the relation. Let σ be an arbitrary signature and let $R \in \sigma$ be a k-ary relation symbol. We show that the class SYM of σ-structures which interprete R by a symmetric relation is strongly uniformly testable on the class $\mathrm{C}_{\sigma,d}$, for each $d \in \mathbb{N}$. Let $\epsilon \in (0,1]$. Given oracle access to a σ-structure $\mathcal{A}$ on n elements, the tester proceeds as follows.

1. Sample $\alpha := \log_{1-\epsilon} \frac{1}{3}$ elements uniformly and independently from the universe $[n]$.
2. For each sampled element i, perform the queries $(R,i,1),\ldots,(R,i,d)$ to obtain all (at most d) tuples $\bar{a}$ of $R^{\mathcal{A}}$ which contain i, and for each such tuple $\bar{a} = (a_1, \ldots, a_k)$ and for each permutation π of $[k]$, check whether the tuple $\pi(\bar{a})$ is also present.
3. If this is true for all α elements, accept. Otherwise reject.

The tester clearly accepts $\mathcal{A}$ if $\mathcal{A} \in$ SYM. Suppose that $\mathcal{A}$ is ϵ-far from SYM. Then there are at least ϵdn bad tuples, i. e. tuples $\bar{a} \in R^{\mathcal{A}}$ such that $\pi(\bar{a}) \notin R^{\mathcal{A}}$ for some permutation π. Since each element of the universe is in at most d tuples, there are at least ϵn different elements that are contained in bad tuples. Hence the probability that a random element is in a bad tuple is at least $\epsilon n/n = \epsilon$. Hence the probability that none of the α sampled elements is in a bad tuple is at most $(1-\epsilon)^{\alpha} = 1/3$. Hence the algorithm accepts with probability at least $2/3$. The tester has query complexity $d \log_{1-\epsilon} \frac{1}{3}$ and constant running time.

4 Monadic second-order logic and bounded tree-width

We are interested in identifying general conditions which ensure the time efficient *uniform testability* of a wide range of properties on relational structures. In this section we consider properties which are definable by sentences of *monadic second order logic with counting* (CMSO) on classes of structures of bounded degree and bounded tree-width.

Before we state the main theorem of this section, we briefly introduce logic. We use the notation which is usual in finite model theory (cf. e.g. [19] for a general overview and [7] for an overview regarding CMSO and the notion of tree-width). Fix a signature σ. An *atomic formula*, is a formula of the form $x = y$ or $R(x_1, \ldots, x_r)$, where $R \in \sigma$ is an r-ary relation symbol and $x, y, x_1, \ldots, x_r$ are (individual) variables. The formulas of first-order logic are built up from atomic formulas using the usual Boolean connectives and existential and universal quantification over the elements of the universe of a structure. The class of all formulas of first-order logic is denoted by FO. *Monadic second-order logic* (MSO) is the extension of first-order logic allowing quantification not only over elements of the universe of a structure, but also over subsets of the universe. Formally, we have two types of variables: *individual variables* (denoted by small letters $x, y, z, x_1, \ldots$), which are interpreted by elements of a structure, and *set variables* (denoted by upper-case letters $X, Y, Z, X_1, \ldots$), which are interpreted by subsets of the universe of a structure. In addition to the atomic

first-order formulas, in MSO we also have atoms Xx saying that x is an element of set X. Furthermore, we have existential and universal quantification over both individual and set variables. CMSO extends MSO by first-order modular counting quantifiers $\exists^m$, for each integer m, where $\exists^m\varphi$ is true in a structure if the number of its elements for which φ is satisfied is divisible by m. A *free variable* of a formula φ is an (individual or set) variable v that does not occur in the scope of a quantifier $\exists v$ or $\forall v$. A *sentence* is a formula without free variables. For a structure $\mathcal{A}$ and a sentence φ we write $\mathcal{A} \models \varphi$ to denote that $\mathcal{A}$ satisfies φ. Detailed introductions can be found in [10, 19].

By $\textsc{Mod}(\varphi)$ we denote the class of all structures satisfying φ. For an arbitrary formula φ and an assignment a in $\mathcal{A}$ to the free variables of φ, we write $(\mathcal{A}, a) \models \varphi$ to denote that $\mathcal{A}$ satisfies φ if the free variables are interpreted according to a. We say that a property P is *definable* in logic $\mathcal{L}$, if there is a sentence φ of $\mathcal{L}$ with $\mathrm{P} = \textsc{Mod}(\varphi)$. For a class C of structures, we say that $\textsc{Mod}(\varphi) \cap \mathrm{C}$ is the property *defined by* φ *on* C.

Recall that we have assumed that the universes of structures are initial segments of the natural numbers. This was used for the definitions surrounding testability. We stress that the linear orders on structures are *not* available in logical formulas.

▶ **Proviso.** *Let $d \in \mathbb{N}$, and fix a finite relational signature σ. From now on, all structures are σ-structures and have degree at most d, unless stated otherwise. Moreover, we let $\alpha := \mathrm{ar}(\sigma)$ and $\mathrm{C} \subseteq \mathrm{C}_{\sigma,d}$.*

We let $\mathrm{C}_t^{\mathrm{tw}}$ denote the class of all σ-structures of degree at most d and tree-width at most t. The main goal of this section is a proof of the following theorem.

▶ **Theorem 5.** *Each property* P *which is CMSO-definable on* $\mathrm{C}_t^{\mathrm{tw}}$ *is uniformly testable on* $\mathrm{C}_t^{\mathrm{tw}}$ *with polylogarithmic time complexity.*

For the proof, we introduce a notion of *locality*, which is based on the distributions of r-discs. In the next section, we will show that locality characterises non-uniform testability. Newman and Sohler's results [21] show that properties of hyperfinite graphs are non-uniformly testable and local. We generalise these to relational structures, and we use the fact that every class of relational structures of bounded tree-width is hyperfinite. This already implies non-uniform testability. We then use semilinearity of $H_r(\mathrm{P})$ for MSO-definable properties on bounded tree-width and a restricted form of ILP to establish polylogarithmic running time.

Since Hanf's paper [16], it is known that properties which are definable by formulas of first-order logic are *local*, in the sense that whether or not a structure has a first-order definable property depends only on the r-discs present in the structure. This is made precise by several notions of locality such as *Hanf locality* and *Gaifman locality* (cf. [17]). These yield a unified combinatorial method for showing that certain properties of sparse structures are not first-order definable. Our notion of (approximate) locality is of similar spirit.

We introduce some notation. We identify each r-histogram vector $\bar{v}$ with n components with a structure $\mathcal{A}$ on n elements over a signature $\sigma_r := \{P_1, \dots, P_{c(r)}\}$ where each relation symbol P_i is unary. The unary relations of $\mathcal{A}$ are pairwise disjoint sets such that $|P_i^{\bar{v}}| := \bar{v}[i]$, for each $i \leq c(r)$. In this way, we can transfer all definitions for structures (e.g., distance, testability) to r-histograms. In particular, note that for all r-histograms $\bar{u}, \bar{v}$ with the same number n of elements, $\bar{u}$ and $\bar{v}$ are ϵ-close iff $\mathrm{dist}(\bar{u}, \bar{v}) \leq \epsilon n$, because histogram vectors are structures of degree $d = 1$. Recall that the ℓ_1-norm of a vector $\bar{v}$ on ℓ components is defined as $\|\bar{u}\|_1 := \sum_{i=1}^{\ell} |\bar{v}[i]|$.

▶ **Lemma 6** (∗)**.** $\mathrm{dist}(\bar{u}, \bar{v}) = \|\bar{u} - \bar{v}\|_1$*, for all r-histograms $\bar{u}, \bar{v}$ with the same number of elements.*

▶ **Definition 7** (Locality)**.** Let $\epsilon \in (0,1]$. A property $\mathrm{P} \subseteq \mathrm{C}$ is *ϵ-local* on C if there exist $r := r(\epsilon) \in \mathbb{N}$ and $\lambda := \lambda(\epsilon) \in (0,1]$ such that for each $\mathcal{A} \in \mathrm{P}$ and $\mathcal{B} \in \mathrm{C}$ with the same number of elements, if $\mathrm{h}_r(\mathcal{A})$ is λ-close to $\mathrm{h}_r(\mathcal{B})$, then $\mathcal{B} \in \epsilon$-close(P).

We call the parameters r and λ the *locality radius* and the *disc proximity* of P for ϵ, respectively. A property is *local* if it is ϵ-local for each $\epsilon \in (0,1]$.

The next example illustrates that locality and testability can indeed be established by very similar arguments.

▶ **Example 8** (Key is local)**.** Recall the definitions of Example 3. We show that Key is local on $\mathrm{C}_{\sigma,d}$. Let $\epsilon \in (0,1]$, let $r := 1$ and $\lambda := \epsilon$. Consider structures $\mathcal{A} \in$ Key and $\mathcal{B} \in \mathrm{C}$ with n elements each and suppose that $\mathrm{h}_r(\mathcal{A})$ and $\mathrm{h}_r(\mathcal{B})$ are λ-close. Each bad tuple of $R^{\mathcal{A}}$ belongs to the 1-disc of its first component. Since $\mathcal{A} \in$ Key, the relation $R^{\mathcal{A}}$ contains no bad tuples, and hence $\mathrm{h}_1(\mathcal{A})[\mathcal{D}] = 0$ for each 1-disc $\mathcal{D}$ such that $R^{\mathcal{D}}$ contains two tuples with the same first component. Since $\mathrm{h}_r(\mathcal{A})$ and $\mathrm{h}_r(\mathcal{B})$ are ϵ-close, the number of elements whose 1-discs contain bad tuples in $\mathcal{B}$ is $\leq \epsilon n$ and each such element is contained in $\leq d$ bad tuples. With $\leq \epsilon d n$ tuple deletions, we obtain a structure $\mathcal{B}' \in$ Key from $\mathcal{B}$. Hence, $\mathcal{B} \in \epsilon$-close(Key).

We now generalise the results of Newman and Sohler [21] to properties on arbitrary hyperfinite classes of relational structures of bounded degree. We begin with some definitions.

A substructure $\mathcal{P}$ of a structure $\mathcal{A}$ on n elements is a *k-partition* of $\mathcal{A}$, if $\mathcal{P}$ and $\mathcal{A}$ have the same universe (i. e. if $P = A$), every connected component of $\mathcal{P}$ contains at most k elements, and for each element $a \in A$, the component $\mathcal{P}[a]$ of a in $\mathcal{P}$ is the substructure of $\mathcal{A}$ induced by its universe $P[a] \subseteq A$. If, furthermore, $\mathrm{dist}(\mathcal{A}, \mathcal{P}) \leq \epsilon n$, we say that $\mathcal{P}$ is a (ϵ, k)-partition of $\mathcal{A}$. A class of structures $\mathrm{C} \subseteq \mathrm{D}$ is *ρ-hyperfinite* on D if for each $\epsilon \in (0,1]$ and each structure $\mathcal{A} \in \mathrm{C}$ there exists a $(\epsilon, \rho(\epsilon))$-partition $\mathcal{P}$ of $\mathcal{A}$ such that $\mathcal{P} \in \mathrm{D}$. We call C *hyperfinite* on D if there exists a function ρ for which C is ρ-hyperfinite on D. This definition generalises the notion of hyperfinite graph classes to general structures.

The proof of Theorem 5 makes use of the following theorem, which can be seen as the generalisation of [21, Theorem 3.1] from graphs to structures. After generalising all ingredients from graphs to relational structures, the proof of Theorem 9 can be put together as in [21, Theorem 3.1].

▶ **Theorem 9** (∗)(Local-Global Theorem)**.** *Let* C *be closed under removing tuples. If* $\mathrm{P} \subseteq \mathrm{C}$ *is hyperfinite on* C*, then* P *is local on* C*.*

We want to approximate the histogram vector of a structure that comes from a hyperfinite class of structures of bounded degree. For this we will make use of Lemma 5.1 in [21], which allows us to approximate the distribution of the r-discs of a graph by looking at a constant number of elements. This lemma easily translates to structures as follows. We write $\mathrm{EstimateFrequencies}_{r,s}$ to denote an algorithm that, given access to a σ-structure $\mathcal{A}$ of degree at most d, samples s elements in A uniformly and independently and explores their r-discs. The algorithm returns the distribution vector $\bar{v}$ of the r-disc-types of this sample.

▶ **Lemma 10.** *Let* $\lambda \in (0,1)$*,* $r \in \mathbb{N}$*. If* $s \geq \frac{c(r)^2}{\lambda^2} \cdot \ln(c(r) + 40)$*, with probability at least* 19/20 *the vector* $\bar{v}$ *returned by* $\mathrm{EstimateFrequencies}_{r,s}$ *on input* $\mathcal{A}$ *satisfies* $\|v - \mathrm{h}_r(\mathcal{A})/|\mathcal{A}|\,\|_1 \leq \lambda$*.*

Our approach to the proof of Theorem 5 can be summarised as follows. It is known that each class of graphs of bounded tree-width (and, more generally, any class of graphs which is minor-closed, cf. [21]) is hyperfinite. From this, it follows that each property P is hyperfinite

on $\mathrm{C}_t^{\mathrm{tw}}$. Hence, by Theorem 9, P is local on $\mathrm{C}_t^{\mathrm{tw}}$. Our aim is to show that a CMSO-definable property P is not only local, but also uniformly testable in polylogarithmic time.

To this end, we study the structure of the sets $\mathrm{H}_r(\mathrm{P})$ for CMSO-definable properties and arbitrary values of r. Using known results from logic, we show that these sets have a particularly simple shape (i.e., they are *semilinear*) if we consider only structures of bounded tree-width. Using this and a result about the complexity of integer linear programming (ILP) with a bounded number of constraints and variables from [11], we can show that there is an algorithm which, on input of an r-histogram on n elements *decides* in polylogarithmic time if $\bar{v}$ is λ-close to the r-histogram of a structure on n elements which belongs to P. This algorithm then, in particular, establishes the testability of $\mathrm{H}_r(\mathrm{P})$, finishing the proof.

The first ingredient to our proof is the following lemma, which carries over from graphs to structures.

▶ **Lemma 11** (∗). *Each property* $\mathrm{P} \subseteq \mathrm{C}_t^{\mathrm{tw}}$ *is hyperfinite on* $\mathrm{C}_t^{\mathrm{tw}}$.

Next, we consider the structure of $\mathrm{H}_r(\mathrm{P})$. For this, we need the following definition.

▶ **Definition 12** (semilinear sets). A set is *semilinear* if it is a finite union of linear sets. A set $M \subseteq \mathbb{N}^c$ is *linear* if $M = \{\bar{v}_0 + a_1\bar{v}_1 + \cdots + a_k\bar{v}_k : a_1, \ldots, a_k \in \mathbb{N}\}$, for $\bar{v}_0, \ldots, \bar{v}_k \in \mathbb{N}^c$.

▶ **Lemma 13.** *For each* $r \in \mathbb{N}$ *and each property* $\mathrm{P} \subseteq \mathrm{C}_t^{\mathrm{tw}}$ *which is CMSO-definable on* $\mathrm{C}_t^{\mathrm{tw}}$, *the set* $\mathrm{H}_r(P)$ *is semilinear.*

Lemma 13 is a corollary to a result of [13] about *many-sorted spectra* of CMSO-sentences.

▶ **Definition 14** (many-sorted spectrum). For every signature σ and each $\ell \in \mathbb{N}$, we let $\sigma_\ell := \sigma \cup \{P_1, \ldots, P_\ell\}$ where $P_1, \ldots P_\ell$ are unary relation symbols which do not occur in σ. A σ_ℓ-structure $\mathcal{A}$ is *ℓ-sorted* if $P_1^{\mathcal{A}}, \ldots, P_\ell^{\mathcal{A}}$ is a partition of A, i.e. $P_1 \cup \cdots \cup P_\ell = A$ and $P_i^{\mathcal{A}} \cap P_j^{\mathcal{A}} = \emptyset$ for all $1 \leq i < j \leq \ell$. We let $n(\mathcal{A}) := (|P_1^{\mathcal{A}}|, \ldots, |P_\ell^{\mathcal{A}}|)$. The *many-sorted spectrum* of a CMSO$[\sigma_\ell]$-sentence φ is the set

$$\mathrm{spec}(\varphi) := \{n(\mathcal{A}) : \mathcal{A} \text{ is a finite } \sigma_\ell\text{-structure}, \mathcal{A} \models \varphi\}.$$

▶ **Theorem 15** ([13]). *If the class defined by a CMSO$[\sigma_\ell]$-sentence φ on the class of all finite σ_ℓ-structures has bounded tree-width, then* $\mathrm{spec}(\varphi)$ *is semilinear.*

Now we can prove our lemma.

Proof of Lemma 13. Let φ be a CMSO$[\sigma]$-sentence defining P on $\mathrm{C}_t^{\mathrm{tw}}$. Let $\tau_1, \ldots, \tau_{c(r)}$ be an enumeration of all r-types according to the fixed ordered on r-types that was used in the definition of $\mathrm{H}_r(\mathrm{P})$. For each $1 \leq i \leq c(r)$, let $\psi_i(x)$ be an FO$[\sigma]$-formula such that for each $\mathcal{A} \in \mathrm{C}_t^{\mathrm{tw}}$ and $a \in A$, $(\mathcal{A}, a) \models \psi_i(x)$ if $(\mathcal{A}, a) \cong \tau_i$. There is an MSO-sentence $\psi_{\mathrm{C}_t^{\mathrm{tw}}}$ which defines the class $\mathrm{C}_t^{\mathrm{tw}}$ on the class of all finite σ-structures (using the forbidden minors for tree-width $\leq t$, see e. g. [8]). Consider the sentence

$$\xi \;:=\; \varphi \wedge \psi_{\mathrm{C}_t^{\mathrm{tw}}} \;\wedge\; \forall x \bigwedge_{1 \leq i \leq c(r)} \big(P_i(x) \leftrightarrow \psi_i(x)\big).$$

Note that $\mathrm{spec}(\xi) = \mathrm{H}_r(\mathrm{P})$ and that all models of ξ have tree-width at most t for some $p := p(t)$. By Theorem 15, we obtain that $\mathrm{H}_r(\mathrm{P})$ is semilinear. ◀

To finish the proof of Theorem 5 it remains to link the semilinearity of $\mathrm{H}_r(\mathrm{P})$ to the testability of P. The following lemma shows that semilinearity of $\mathrm{H}_r(\mathrm{P})$ implies decidability of ϵ-close$(\mathrm{H}_r(\mathrm{P}))$ with polylogarithmic running time.

▶ **Lemma 16** (∗) (Approximate realisability). *Let $r \in \mathbb{N}$ and $\epsilon \in \mathbb{Q} \cap (0,1]$, and let $M \subseteq \mathbb{N}^{c(r)}$ be a semilinear set of r-histograms. Then there is an algorithm that, given an r-histogram $\bar{u} \in \mathbb{N}^{c(r)}$ on n elements (in binary encoding), decides whether $\bar{u} \in \epsilon\text{-close}(M)$ in time polylogarithmic in n.*

For the proof of Lemma 16, we phrase the conditions for belonging to ϵ-close(M) as an ILP with a constant number of variables and constraints. Using results of [18], [11], we obtain the desired running time. It remains to prove the following lemma.

Proof of Theorem 5. Let P be a property defined by some fixed CMSO-formula on $\mathrm{C}_t^{\mathrm{tw}}$. Given ϵ, we construct an ϵ-tester for P for inputs $\mathcal{A} \in \mathrm{C}_t^{\mathrm{tw}}$ on n elements. By Lemma 11, $\mathrm{C}_t^{\mathrm{tw}}$ is hyperfinite, and hence, by Theorem 9, P is local on $\mathrm{C}_t^{\mathrm{tw}}$. Let $r = r(\epsilon)$ be the locality radius and $\lambda = \lambda(\epsilon)$ the disc proximity of P for ϵ, as in the definition of locality. Pick a sample size s for r and $\lambda/2$ as required for the algorithm EstimateFrequencies$_{r,s}$ of Lemma 10, and run the algorithm to obtain an approximation $\bar{v}$ of the frequency vector of $\mathcal{A}$ with high probability. Accept, if $n \cdot \bar{v}$ is $\lambda/2$-close to $H_r(\mathrm{P})$, and reject otherwise (using the algorithm of Lemma 16). It is easy to see that the algorithm is correct. Since EstimateFrequencies$_{r,s}$ runs in constant time, and the algorithm of Lemma 16 runs in time polylogarithmic in n, we obtain uniform testability with polylogarithmic running time. ◀

We remark that the vectors spanning the semilinear set $\mathrm{H}_r(\mathrm{P})$ (in the proof of Theorem 5) can be computed from the CMSO-definition of P. This is implicit in [13].

The same argument as in the proof of Theorem 5 can be used to show the following lemma.

▶ **Lemma 17** (∗). *If a property $\mathrm{P} \subseteq \mathrm{C}$ is local and ϵ-close$_{\mathrm{H}_r(\mathrm{C})}(\mathrm{H}_r(\mathrm{P}))$ is decidable in time* polylog(n)*, for each $\epsilon \in (0,1]$ and $r \in \mathbb{N}$, then P is uniformly testable on C in time* polylog(n)*.*

For the proof of Lemma 17, we first approximate the distribution of r-types by sampling. Then we accept if this distribution is sufficiently close to the distribution of a structure in P on the same number of elements as the input structure.

5 Locality and testability

In this section we characterise non-uniform testability by locality. Inspired by the proof for uniform testability of MSO on bounded tree-width, we introduce the notion of *effective locality*, which adds the requirement that a certain realisability problem for the neighbourhood distributions be solvable. We use effective locality to characterise uniform testabilty (on decidable classes). We believe that the characterisations are interesting, as they provide a purely structural criterion for testability and non-testability. While it is implicitely clear that non-uniform testability 'only depends on the local neighbourhoods', to our knowledge this has not been cast into a characerisation in the literature so far. Uniform testability has not been characterised before. Furthermore, they explain the role of uniformity which has been brought up by the general results on non-uniform testability of Newman, Sohler [21]. In this section, with a few exceptions, we disregard the running times of testers since we are interested in the structural properties of testability.

The first main theorem of this section shows that non-uniform testability is equivalent to locality.

▶ **Theorem 18** (Locality). *Then for every property $\mathrm{P} \subseteq \mathrm{C}$, P is non-uniformly testable on C if, and only if, P is local on C.*

We mention that from Theorems 18 and 9 it follows that every property of hyperfinite databases is testable. This is a generalisation of [21].

▶ **Corollary 19.** *Let* C *to be closed under removing tuples. If* C *is hyperfinite, then every property* $\mathrm{P} \subseteq \mathrm{C}$ *is non-uniformly testable on* C*.*

For example, Theorem 18 can be used for purely graph-theoretic proofs of non-testability (e.g. in the proof that bipartiteness is not testable with a constant number of queries [15]). Note that locality of a property P is not enough to ensure *uniform testability*. This can be easily shown since the halting problem for turing machines with empty input can be trivially encoded as a local property of graphs of degree 0. Intuitively, we would like to use the locality of P to construct a tester for P as follows: on input of a structure $\mathcal{A}$, (1) approximate the distribution of the r-types by random sampling and (2) accept $\mathcal{A}$ if this distribution is sufficiently close to the distribution of some structure from P. The notion of locality does not guarantee that step (2) can be implemented effectively, which motivates introducing *effective locality*.

We now introduce *effective locality*. Recall from above that our notion of testability applies, in particular, to properties of r-histograms which we treat as structures of degree 1.

▶ **Definition 20.** $\mathrm{P} \subseteq \mathrm{C}$ is *effectively local* on C if it is local on C and for each $\epsilon \in (0,1)$ and for the corresponding locality radius $r := r(\epsilon)$, the problem $\mathrm{H}_r(\mathrm{P})$ is uniformly testable on $\mathrm{H}_r(\mathrm{C})$. If the running time of the tester is $T(n)$, we say that P is $T(n)$*-effectively local*.

The *realisability problem* (cf. e. g. [1]) for a graph parameter f which maps graphs G to $f(G) \in \mathbb{N}^k$ and for a class C of graphs is the decision problem which, on input of a vector $\bar{v} \in \mathbb{N}^k$, asks if there exists a graph $G \in \mathrm{C}$ such that $f(G) = \bar{v}$. Hence, the problem $\mathrm{H}_r(\mathrm{P})$ can be viewed as a *realisability problem* for structures.

We show that effective locality characterises uniform testability. In the following theorem, we need the notion of a *promise problem*. Following e.g. [14], we define this as a pair of disjoint languages $(L_{\mathrm{YES}}, L_{\mathrm{NO}})$ of binary strings. We say that $(L_{\mathrm{YES}}, L_{\mathrm{NO}})$ is *solvable* if there exists an algorithm which accepts all inputs which belong to L_{YES} and rejects all inputs which belong to L_{NO}. Note that a brute-force derandomisation of an ϵ-tester for a property P yields a deterministic algorithm which solves the promise problem $(\mathrm{P}, \epsilon\text{-far}(\mathrm{P}))$. From a conceptual point of view, disregarding running times, it is more convenient to consider the promise problem.

The second main theorem of this section characterises uniform testability.

▶ **Theorem 21** (Effective Locality)**.** *Let* C *be a decidable class of structures, that is closed under removing tuples. For each property* $\mathrm{P} \subseteq \mathrm{C}$*, the following statements are equivalent.*

1. P *is uniformly testable on* C*.*
2. P *is effectively local on* C*.*
3. P *is local and the promise problem* $(\mathrm{P}, \epsilon\text{-far}(\mathrm{P}))$ *is solvable for each* $\epsilon \in (0,1]$*.*

We break the proofs of Theorem 18 and Theorem 21 into several lemmas.

▶ **Lemma 22** (∗)**.** *Let* C *be a decidable class of* σ*-structures. Let* $\mathrm{P} \subseteq \mathrm{C}$ *be a property such that the promise problem* $(\mathrm{P}, \epsilon\text{-far}(\mathrm{P}))$ *is solvable for each* $\epsilon \in (0,1]$*. Then the promise problem* $\big(\mathrm{H}_r(\mathrm{P}), \lambda\text{-far}(\mathrm{H}_r(\mathrm{P}))\big)$ *is solvable for each* $r \in \mathbb{N}$ *and* $\lambda \in (0,1]$*.*

▶ **Lemma 23** (∗)**.** *Let* C *be a decidable class of* σ*-structures. Let* $\mathrm{P} \subseteq \mathrm{C}$ *be a local property such that the promise problem* $\big(\mathrm{H}_r(\mathrm{P}), \lambda\text{-far}(\mathrm{H}_r(\mathrm{P}))\big)$ *is solvable for each* $r \in \mathbb{N}$ *and* $\lambda \in (0,1]$*, then* P *is uniformly testable.*

Without the solvability of the promise problem, we obtain non-uniform testability instead of uniform testability.

Now we prove the implication from effective locality to uniform testability of Theorem 21. The following lemma is stronger than what is needed here, because it also takes the running time of the tester into account. It can be seen as a generalisation of Lemma 17.

▶ **Lemma 24** (∗). *If a property* $\mathrm{P} \subseteq \mathrm{C}$ *is effectively* $\mathrm{polylog}(n)$*-local on* C*, then it is uniformly testable on* C *in time* $\mathrm{polylog}(n)$*.*

▶ **Corollary 25** (∗). *If a property* P *is local on* C*, then it is non-uniformly testable on* C*.*

Proof sketch. Since $\mathrm{P}|n$ is finite and local, it is effectively local and the result follows from Lemma 24. ◀

▶ **Lemma 26** (∗). *If a property* $\mathrm{P} \subseteq \mathrm{C}$ *is non-uniformly testable on* C*, then* P *is local on* C*.*

Proof idea. It is intuitively clear that a tester with constant query complexity can only inspect discs of a constant radius r. This is made precise by the Canonical Tester Lemma which was proved by Czumaj, Peng, Sohler in the context of property testing for directed graphs [9, Lemma 3.1]. This lemma extends to relational structures. It can be shown that, on structures with sufficiently close r-histograms, a tester for P will see the same r-discs with high probability. This is done in a similar way as in the proof of the "local versus global graph structure"-theorem of Newman, Sohler [21, Theorem 3.1]. Hence, the tester will not be able to distinguish between structures with close r-histograms, so if one of the structures is in P, the other has to be in ϵ-close(P). This yields locality of P. ◀

We can now finish the proofs of both characterisation theorems.

Proof of Theorem 18. The theorem follows from Corollary 25 and Lemma 26. ◀

Proof of Theorem 21. The implication from statement 1 (uniform testability) to statement 3 (locality and solvability of $(\mathrm{P}, \epsilon\text{-far}(\mathrm{P}))$) follows from Theorem 18 (locality) and by derandomisation (solvability). The implication from 3 to 2 (testability of $\mathrm{H}_r(\mathrm{P})$) is established by Lemma 22. The implication from 2 to 1 is established by Lemma 23. ◀

6 Conclusion

We introduced property testing for relational databases of bounded degree. Our main result is a logical meta-theorem proving testability of CMSO with constant query complexity and polylogarithmic running time for databases of bounded tree-width, and we provide characterisations of testability and uniform testability in the model. Our tester for CMSO has two-sided error (because it samples the distribution of the r-discs), and it would be interesting to know if a one-sided error can be achieved.

Since monadic second-order logic on words characterises the regular languages, Theorem 5 shows in particular, that regular languages are testable with a constant number of queries in our model. In [2] testability of regular languages was already shown for a more restrictive model, based on Hamming distance. Similarly, our result implies that regular (ranked) tree languages are testable with a constant number of queries. In [20], testability of tree languages was shown in a different model, using tree edit distance with an additional operation called *moves*. The question of [5] (explicitly stated in [20]) whether regular tree languages with Hamming distance are testable, remains open.

A further interesting question is whether all properties definable in first-order logic are (uniformly) testable is left open, and we are currently working on this. Furthermore, obtaining a logical characterisation of the (uniformly) testable properties is interesting and challenging open problem. It would also be interesting to determine the precise relation between our notion of locality and Hanf-locality. Being reminiscent of realisability of degree sequences of graphs, realisability of histograms seems worth studying in more detail.

References

1. Martin Aigner and Eberhard Triesch. Realizability and uniqueness in graphs. *Discrete Mathematics*, 136(1-3):3–20, 1994. doi:10.1016/0012-365X(94)00104-Q.
2. Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular languages are testable with a constant number of queries. *SIAM J. Comput.*, 30(6):1842–1862, 2000. doi:10.1137/S0097539700366528.
3. Michael A. Bender and Dana Ron. Testing properties of directed graphs: acyclicity and connectivity. *Random Struct. Algorithms*, 20(2):184–205, 2002. doi:10.1002/rsa.10023.
4. Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. Approximate query processing: No silver bullet. In Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 511–519. ACM, 2017. doi:10.1145/3035918.3056097.
5. Hana Chockler and Orna Kupferman. w-regular languages are testable with a constant number of queries. *Theor. Comput. Sci.*, 329(1-3):71–92, 2004. doi:10.1016/j.tcs.2004.08.004.
6. B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science (Vol. B)*, pages 193–242. MIT Press, Cambridge, MA, USA, 1990. URL: http://dl.acm.org/citation.cfm?id=114891.114896.
7. B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2012.
8. B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2012.
9. Artur Czumaj, Pan Peng, and Christian Sohler. Relating two property testing models for bounded degree directed graphs. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 1033–1045. ACM, 2016. doi:10.1145/2897518.2897575.
10. H.-D. Ebbinghaus and J. Flum. *Finite model theory*. Springer, 2005.
11. Friedrich Eisenbrand. Fast integer programming in fixed dimension. In Giuseppe Di Battista and Uri Zwick, editors, *Algorithms - ESA 2003, 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003, Proceedings*, volume 2832 of *Lecture Notes in Computer Science*, pages 196–207. Springer, 2003. doi:10.1007/978-3-540-39658-1_20.
12. Wenfei Fan, Floris Geerts, and Leonid Libkin. On scale independence for querying big data. In Richard Hull and Martin Grohe, editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 51–62. ACM, 2014. doi:10.1145/2594538.2594551.
13. Eldar Fischer and Johann A. Makowsky. On spectra of sentences of monadic second order logic with counting. *J. Symb. Log.*, 69(3):617–640, 2004. doi:10.2178/jsl/1096901758.

14 Oded Goldreich. On promise problems: A survey. In Oded Goldreich, Arnold L. Rosenberg, and Alan L. Selman, editors, *Theoretical Computer Science, Essays in Memory of Shimon Even*, volume 3895 of *Lecture Notes in Computer Science*, pages 254–290. Springer, 2006. doi:10.1007/11685654_12.

15 Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002. doi:10.1007/s00453-001-0078-7.

16 W. Hanf. *The Theory of Models*, chapter Model-theoretic methods in the study of elementary logic, pages 132–145. North Holland, 1965.

17 L. Hella, L. Libkin, and J. Nurmonen. Notions of locality and their logical characterizations over finite models. *Journal of Symbolic Logic*, 64(4):1751–1773, 1999.

18 H. W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research.*, 8(4):538–548, 1983.

19 L. Libkin. *Elements Of Finite Model Theory (Texts in Theoretical Computer Science. An Eatcs Series)*. Springer, 2004.

20 Frédéric Magniez and Michel de Rougemont. Property testing of regular tree languages. *Algorithmica*, 49(2):127–146, 2007. doi:10.1007/s00453-007-9028-3.

21 I. Newman and C. Sohler. Every property of hyperfinite graphs is testable. *SIAM Journal on Computing*, 42(3):1095–1112, 2013. Conference version published at STOC 2011.

22 Rohit Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966. doi:10.1145/321356.321364.

Erdős-Pósa Property of Obstructions to Interval Graphs

Akanksha Agrawal
University of Bergen, Bergen, Norway
akanksha.agrawal@ii.uib.no

Daniel Lokshtanov
University of Bergen, Bergen, Norway
daniello@ii.uib.no

Pranabendu Misra
University of Bergen, Bergen, Norway
pranabendu.misra@ii.uib.no

Saket Saurabh
University of Bergen, Bergen, Norway and
Institute of Mathematical Sciences, HBNI, Chennai, India and UMI ReLax
saket@imsc.res.in

Meirav Zehavi
University of Bergen, Bergen, Norway
meirav.zehavi@ii.uib.no

Abstract

The duality between packing and covering problems lies at the heart of fundamental combinatorial proofs, as well as well-known algorithmic methods such as the primal-dual method for approximation and win/win-approach for parameterized analysis. The very essence of this duality is encompassed by a well-known property called the Erdős-Pósa property, which has been extensively studied for over five decades. Informally, we say that a class of graphs $\mathcal{F}$ admits the Erdős-Pósa property if there exists f such that for any graph G, either G has k vertex-disjoint "copies" of the graphs in $\mathcal{F}$, or there is a set $S \subseteq V(G)$ of $f(k)$ vertices that intersects all copies of the graphs in $\mathcal{F}$. In the context of any graph class $\mathcal{G}$, the most natural question that arises in this regard is as follows – do obstructions to $\mathcal{G}$ have the Erdős-Pósa property? Having this view in mind, we focus on the class of interval graphs. Structural properties of interval graphs are intensively studied, also as they lead to the design of polynomial-time algorithms for classic problems that are NP-hard on general graphs. Nevertheless, about one of the most basic properties of such graphs, namely, the Erdős-Pósa property, nothing is known. In this paper, we settle this anomaly: we prove that the family of obstructions to interval graphs – namely, the family of chordless cycles and ATs – admits the Erdős-Pósa property. Our main theorem immediately results in an algorithm to decide whether an input graph G has k vertex-disjoint ATs and chordless cycles, or there exists a set of $\mathcal{O}(k^2 \log k)$ vertices in G that hits all ATs and chordless cycles.

2012 ACM Subject Classification Mathematics of computing → Extremal graph theory, Mathematics of computing → Graph algorithms

Keywords and phrases Interval Graphs, Obstructions, Erdős-Pósa Property

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.7

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 7; pp. 7:1–7:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Packing and covering problems are ubiquitous in both graph theory and computer science. The duality between packing and covering problems lies at the heart of not only fundamental combinatorial proofs, but also well-known algorithmic methods such as the primal-dual method for approximation and win/win-approach for parameterized analysis. The very essence of this duality is encompassed by a well-known property called the Erdős-Pósa property. This property, being both simple and powerful, has been extensively studied for over five decades. In the context of any graph class $\mathcal{G}$, the most natural question that arises in this regard is as follows – do obstructions to $\mathcal{G}$ have the Erdős-Pósa property? Having this view in mind, we focus on the class of interval graphs. Arguably, this is the most basic class of graphs that can be viewed as geometric inputs – indeed, an interval graph is the intersection graph of a family of intervals on the real lines. Interval graphs are among the most well-studied classes of graphs in the literature. In particular, the usage of interval graphs as models is relevant to a wide variety of applications, ranging from resource allocation in operations research and scheduling theory to assembling contiguous subsequences in DNA mapping. From an algorithmic point of view, the structural properties of interval graphs are also intensively studied as they allow to design polynomial-time algorithms for well-known problems in computer science, such as Independent Set and Hamiltonian Path, that are NP-hard on general graphs. Nevertheless, about one of the most basic properties of such graphs, namely, the Erdős-Pósa property, nothing is known! Our main contribution settles this anomaly: we prove that obstructions to interval graphs admit the Erdős-Pósa property.

Before we turn to consider our contribution in more detail, we present a gentle introduction to the rich realm of studies of Erdős-Pósa properties. For this purpose, we first define packing and covering problems. Let $\preceq$ be a containment relation (of a graph into another graph), and let $\mathcal{F}$ be a family of graphs. For example, we can define the containment relationship $\preceq$ as follows: for graphs G and H, $H \preceq G$ if and only if H is an induced subgraph/subgraph/minor/topological minor of G. In this setting, $(\mathcal{F}, \preceq)$-Packing is the problem whose input consists of a graph G and an integer k, and the objective is to decide if G has k vertex-disjoint subsets, $S_1, S_2, \ldots, S_k \subseteq V(G)$, where for each $i \in [k]$, there exists $F \in \mathcal{F}$ such that $F \preceq G[S_i]$. For example, if $\mathcal{F} = \{F\}$ and the relation refers to induced subgraphs, then we simply ask whether G has k vertex-disjoint "exact copies" of F. The $(\mathcal{F}, \preceq)$-Covering problem has the same input, but its objective is to decide if there is a set $S \subseteq V(G)$ of size at most k such that there does not exist $F \in \mathcal{F}$ that satisfies $F \preceq G - S$. Some well-known examples of packing problems (and their corresponding covering problems) are Maximum Matching (Vertex Cover), Vertex-Disjoint s-t Paths (s-t-Separator), Cycle Packing (Feedback Vertex Set), P_3-Packing (Cluster Vertex Deletion), and Triangle Packing (Triangle Free Deletion).

Kőnig's and Menger's theorems are cornerstones of Graph Theory in general, and of the study of packing and covering problems in particular, which have also found a wide variety of applications in computer science. For example, Menger's theorem is particularly relevant to survivable network design (see, e.g., [5, 46]) and combinatorial optimization (see, e.g., [43, 19]). Formally, Kőnig's theorem states that in bipartite graphs, the maximum size of a matching *equals* the minimum size of a vertex cover [30, 13]. Menger's theorem also exhibits an equality – it states that for a given graph G and a pair of vertices s and t, either G has k vertex-disjoint paths between s and t or there is a set $S \subseteq V(G) \setminus \{s, t\}$ of size k such that $G - S$ has no path between s and t [33, 13]. Both theorems relate a packing problem to

a covering problem[1], by exhibiting equality between the size of a maximum packing and the size of a minimum covering. However, most natural packing and covering problems are not known to exhibit such an equality; in fact, frequently such an equality is *proven* not to exist. By relaxing the notion of equality, we enter the rich realm of the Erdős-Pósa properties.

The Erdős-Pósa Property. A celebrated theorem by Erdős and Pósa [14] states that for any graph G, either there is a set of k vertex-disjoint cycles in G, or there is a set $S \subseteq V(G)$ of $f(k) = \mathcal{O}(k \log k)$ vertices that intersects (covers) all cycles of G.[2] Notably, Erdős and Pósa [14] also showed that there exists a constant c and infinitely many pairs (G, k) such that G has neither k vertex-disjoint cycles nor a set $S \subseteq V(G)$ of $ck \log k$ vertices that covers all cycles of G. That is, not only equality cannot be expected, but also any function $f(k) = o(k \log k)$. We remark that later, Simonovits [45] provided concrete examples which realize the lower bound. The result of Erdős and Pósa [14] initiated a flurry of extensive study of the so called "Erdős-Pósa property" for various families of graphs as well as containment relationships. Formally, a family of graphs $\mathcal{F}$ and a containment relation $\preceq$ are said to admit the Erdős-Pósa property if there exists a function $f(\cdot)$ such that given a graph G and an integer k, either there are k vertex-disjoint subsets $S_1, \ldots S_k \subseteq V(G)$ so that for each $i \in [k]$, there is $F \in \mathcal{F}$ satisfying $F \preceq G[S_i]$, or there is a set $S \subseteq V(G)$ of size at most $f(k)$ so that there is no $F \in \mathcal{F}$ satisfying $F \preceq G - S$. Here, the first question that comes to mind is – do all families of graphs $\mathcal{F}$ and containment relationships $\preceq$ exhibit the Erdős-Pósa property?

The answer to this question is negative. For example, consider a fixed graph H, and let $\mathcal{F}(H)$ be the family of graphs that contain H as a minor. Robertson and Seymour [42] showed that $\mathcal{F}(H)$ with the containment relation referring to subgraphs admits the Erdős-Pósa property if and only if H is a planar graph. This result generalizes the result in [14]. However, the function $f(\cdot)$ given by [42] is exponential – can it be made polynomial? A few years ago, the bound was improved to $\mathcal{O}(k \log^c k)$ by Chekuri and Chuzhoy [12] following a more general approach which is applicable to other families as well. A well-known example of a different flavor conerns odd cycles. Specifically, for $\mathcal{F}$ being the family of odd length cycles, Reed [39] showed that $\mathcal{F}$ (for subgraphs and induced subgraphs) does not admit the Erdős-Pósa property.

Since the emergence of the result of Erdős and Pósa [14], a multitude of studies on the Erdős-Pósa property have appeared in the literature for several combinatorial objects beyond graphs. This includes extensions to digraphs [32, 44, 40, 22, 20], rooted graphs [9, 26, 35, 24], labeled graphs [29], signed graphs [23, 3], hypergraphs [1, 6, 7], matroids [16], helly-type theorems [21], H-minors [41], H-immersions [17, 31], and H-butterfly directed minors [2] (also see [38]). This list is not comprehensive but rather illustrative. We refer to surveys such as [37] for more information. Even for subfamilies of cycles alone, there is a vast literature devoted to the Erdős-Pósa property. Studies of the Erdős-Pósa property for subfamilies of cycles include, for example, long cycles (subgraphs) [4, 34], directed cycles (subgraphs and induced subgraphs) [40, 20], chordless cycles (induced subgraphs) [25] and cycles intersecting a prescribed vertex set [27, 35]. Not all subfamiles of cycles admit the Erdős-Pósa property. For example, recall the result stated earlier regarding the family of odd cycles [39]. For this subfamily of cycles alone, there has been a sequence of research about finding classes of

[1] For example, Kőnig's theorem addresses the class $\mathcal{F} = \{F\}$ such that F is the graph on a single edge, where $\preceq$ refers to induced subgraphs/subgraphs.

[2] In the terminology of packing and covering, we address the class $\mathcal{F}$ of all cycles, where $\preceq$ refers to induced subgraphs/subgraphs.

graphs for which the family of odd cycles (subgraphs and induced subgraphs) admits the Erdős-Pósa property. This includes planar graphs [15], or graphs with certain connectivity constraints [48, 36, 28, 24]. Not only the family of odd cycles does not admit the Erdős-Pósa property, but also subfamilies such as the family of chordless cycles of length at least 5 [25].

A large number of the results above can be viewed as the question of packing or covering obstructions to a class of graphs. In some of these papers, this view is explicitly stated as the motivation behind the conducted studies. For example, the classic result by Erdős and Pósa [14] regards the question of packing and covering obstructions to forests. The results concerning odd cycles address obstructions to bipartite graphs. The setting of the work about packing and covering chordless cycles, as presented by [25], addresses obstructions to chordal graphs. Furthermore, Kőnig's theorem relates to obstructions to edgeless graphs, and the work by Robertson and Seymour [42] relates to obstructions to subfamilies of minor free graphs. We remark that other results can also be interpreted in this manner. Given that the class of interval graphs is among the most basic, well-studied families of graphs, we find it important to study the Erdős-Pósa [14] property with respect to it. Let $\mathcal{F}$ be the family of chordless cycles and asteroidal triples (ATs), see Section 2. It is well known that the class of interval graphs is precisely the class of graphs that exclude every graph in $\mathcal{F}$ as an induced subgraph [18, 8]. Given this clean characterization, the following question naturally arises:

> Does the family of chordless cycles and ATs – that is, obstructions to interval graphs – admit the Erdős-Pósa property?

Our Contribution. We provide an affirmative answer to the question above. Moreover, the dependency of the size of the covering set on k in our result is only $\mathcal{O}(k^2 \log k)$.[3] Specifically, we obtain the following theorem, where from now on, "obstructions" refer to ATs and chordless cycles.

▶ **Theorem 1.** *Let G be a graph, and let $k \in \mathbb{N}$. At least one of the following conditions holds:* **(i)** *G has k vertex-disjoint obstructions;* **(ii)** *there exists a subset $D \subseteq V(G)$ of size $\mathcal{O}(k^2 \log k)$ such that $G - D$ is an interval graph.*

As a consequence of our main theorem, we also derive an algorithm to decide whether an input graph G has k vertex-disjoint obstructions (to interval graphs), or there exists a set of $\mathcal{O}(k^2 \log k)$ vertices in G that hits all such obstructions.

We conclude the introduction with a high-level (informal) overview of our proof. We begin by easily "getting rid" of all chordless cycles due to the work by [25], as well as all small ATs. Now, the heart of our proof consists of two main components. First, we exhibit the Erdős-Pósa property of the family of ATs on graphs that have a clique caterpillar (that is, a tree decomposition that is a caterpillar, where every bag is a clique). Second, we show how this result can be utilized to derive our main theorem by analyzing "conflict-free sets" (defined below) with respect to a modular tree decomposition of the graph. Let us now elaborate on each component.

To analyze the case of a clique caterpillar, we present a procedure that at each iteration, finds an AT $\mathbb{O}$ with specific properties, inserts a set S of $\mathcal{O}(k)$ new vertices into a set $S^\star$ initialized to be empty, and removes the vertices in S from the graph (only for the sake of the execution of the procedure). Specifically, the set S consists of the terminals, centers and

[3] In fact, all of our arguments achieve the dependency $\mathcal{O}(k^2)$, but we gain an extra $\log k$ factor due to an invocation of a result by Kwon and Kim [25].

a few base vertices of $\mathbb{O}$, as well as all of the vertices of a "small" separator between the non-shallow terminals of $\mathbb{O}$ that we push as much as possible to the right of the caterpillar. The procedure terminates once the graph becomes an interval graph. Hence, it is clear that if at most $\mathcal{O}(k)$ iterations take place, then $S^\star$ is a set of size $\mathcal{O}(k^2)$ that intersects all ATs, which implies that our job is done. Otherwise, we require an intricate analysis to establish the existence of k vertex-disjoint ATs. Roughly, the two main components here are (1) from the sequence of ATs encountered by our procedure, we can extract a sequence of the same length (of possibly *different* ATs) where each AT has the property that the subpath of its base that lies after the separator does not intersect any AT positioned after it in the sequence, and (2) from the modified sequence, we can extract a *subsequence* of ATs where disjointess is also guaranteed with respect to base vertices that lie before the separator.

Towards the proof of the second item, we first show that for every sequence "resembling" the one encountered by our procedure, and for all ATs $\mathbb{O}$ and $\mathbb{O}'$ in that sequence such that $\mathbb{O}'$ comes before $\mathbb{O}$, we have the following property: only the leftmost terminal and base vertex of $\mathbb{O}$ can belong to the base path of $\mathbb{O}'$ that lies before the separator associated with $\mathbb{O}'$, and even that is only possible under certain conditions. This result then allows us to further argue about the relation between every *three* ATs in the sequence with respect to the "left sides of separators". Having established this relation, the argument about a complete sequence is derived. Towards the proof of the first item, we first show that for any AT $\mathbb{O}$ in the sequence, we can find a path between a vertex in the separator associated with $\mathbb{O}$ and the right terminal of $\mathbb{O}$ that avoids all ATs coming after $\mathbb{O}$ in the sequence. Then, by relying on structural results by Cao and Marx [11], we argue that this path can be used to replace part of $\mathbb{O}$ so that the result is yet another AT.

Let us now turn to our analysis of the general case – specifically, we explain how it is reduced to instances of the case of a clique caterpillar. We define "problematic" nodes in the modular tree decomposition of the input graph as the nodes associated with subgraphs that contain at least one AT that is not present in any of the subgraphs associated with their children. This definition also immediately gives rise to an association between nodes and ATs, so that each AT is associated with exactly one node. We observe that maximal modules of problematic nodes are vertex disjoint, and that each problematic node has "many" children. It is also easily shown that the set of all problematic nodes can be partitioned into two sets that have no "conflict" – that is, on the unique path between every two nodes of one set, there exists a node of the other set. The point in analyzing each conflict-free set P separately is that for each problematic node in such a set, we prove that there exist at least k vertices in the subgraph associated with that node that do not belong to any subgraph associated with its problematic descendants from P. In particular, this allows us to examine each problematic node *individually*, and associate an instance of the clique caterpillar case with it (the construction of the caterpillar decomposition itself partially follows from structural results by Cao and Marx [11]). Specifically, we are able to collect the sets of ATs found in each instance, and argue that (after some modification) all of these ATs across all the sets are in fact vertex disjoint. This result then allows us to handle the "packing perspective" of the proof. We remark that although we can create $\mathcal{O}(k)$ instances of the clique caterpillar case, and each individual instance can create a gap of $\mathcal{O}(k^2)$, we eventually get a gap of only $\mathcal{O}(k^2)$ rather than $\mathcal{O}(k^3)$ as we argue that the sum of the contributions to the gap of all individual instances is $\mathcal{O}(k^2)$.

Due to lack of space, proofs of statements marked by "*" were omitted.

2 Preliminaries

For $n \in \mathbb{N}$, we use $[n]$ as a shorthand for $\{1, 2, \ldots, n\}$. Given a function $f : A \to B$ and a subset $A' \subset A$, we use $f|_{A'}$ to denote the restriction of f to A'.

We refer to standard terminology from the book of Diestel [13] for graph-related terms that are not explicitly defined here. When the graph G is clear from context, denote $n = |V(G)|$ and $m = |E(G)|$. We say that a vertex v in G is *simplicial* if $N_G(v)$ induces a clique. A *caterpillar* is a tree T for which there exists a subpath P of T, called a *central path*, such that the removal of the vertices of P from T results in an edgeless graph. Given a rooted tree T and a vertex $v \in V(T)$, we use $T|_v$ to denote the subtree of T rooted at v. Moreover, $\mathsf{child}(v)$ denotes the set of children of v in T. We do not treat a vertex as a descendant of itself. A *chordal graph* is a graph with no chordless cycle on at least four vertices.

Interval Graphs. An *interval graph* is a graph G that does not contain any of the following graphs, called *obstructions*, as an induced subgraph.

- **Long Claw.** A graph $\mathbb{O}$ such that $V(\mathbb{O}) = \{t_\ell, t_r, t, c, b_1, b_2, b_3\}$ and $E(\mathbb{O}) = \{\{t_\ell, b_1\}, \{t_r, b_3\}, \{t, b_2\}, \{c, b_1\}, \{c, b_2\}, \{c, b_3\}\}$.
- **Whipping Top (or Umbrella).** A graph $\mathbb{O}$ such that $V(\mathbb{O}) = \{t_\ell, t_r, t, c, b_1, b_2, b_3\}$ and $E(\mathbb{O}) = \{\{t_\ell, b_1\}, \{t_r, b_2\}, \{c, t\}, \{c, b_1\}, \{c, b_2\}, \{b_3, t_\ell\}, \{b_3, b_1\}, \{b_3, c\}, \{b_3, b_2\}, \{b_3, t_r\}\}$.
- **†-AW.** A graph $\mathbb{O}$ such that $V(\mathbb{O}) = \{t_\ell, t_r, t, c\} \cup \{b_1, b_2, \ldots, b_z\}$, where $t_\ell = b_0$ and $t_r = b_{z+1}$, $E(\mathbb{O}) = \{\{t, c), \{t_\ell, b_1\}, \{t_r, b_z\}\} \cup \{\{c, b_i\} \mid i \in [z]\} \cup \{\{b_i, b_{i+1}\} \mid i \in [z-1]\}$, and $z \geq 2$. A †-AW where $z = 2$ is called a *net*.
- **‡-AW.** A graph $\mathbb{O}$ such that $V(\mathbb{O}) = \{t_\ell, t_r, t, c_1, c_2\} \cup \{b_1, b_2, \ldots, b_z\}$, where $t_\ell = b_0$ and $t_r = b_{z+1}$, $E(\mathbb{O}) = \{\{t, c_1\}, \{t, c_2\}, \{c_1, c_2\}, \{t_\ell, b_1\}, \{t_r, b_z\}, \{t_\ell, c_1\}, \{t_r, c_2\}\} \cup \{\{c, b_i\} \mid i \in [z]\} \cup \{\{b_i, b_{i+1}\} \mid i \in [z-1]\}$, and $z \geq 1$. A ‡-AW where $z = 1$ is called a *tent*.
- **Hole.** A chordless cycles on at least four vertices.

Long claws and whipping tops are also called ATs, but we shall reserve this name for †-AWs and ‡-AWs (AW stand for Asteroidal Witness).[4] An obstruction $\mathbb{O}$ is *minimal* if there does not exist an obstruction $\mathbb{O}'$ such that $V(\mathbb{O}') \subset V(\mathbb{O})$. In each of the first four obstructions, the vertices t_ℓ, t_r, and t are called *terminals*, the vertices c, c_1, and c_2 are called *centers*, and the other vertices are called *base vertices*. To simplify notation, when we consider a †-AW, we use c_1 and c_2 to refer to c (this allows us to refer to a †-AW and a ‡-AW in a unified manner). Furthermore, the vertex t is called the *shallow terminal*. The induced path on the set of base vertices is called the *base* of the AT, and it is denoted by $\mathsf{base}(\mathbb{O})$. Moreover, we say that the induced path on the set of base vertices, t_ℓ and t_r is the *extended base* of the AT, and it is denoted by $P(\mathbb{O})$. Given a graph G, a vertex v is *shallow in* G if G has at least one AT where v is the shallow terminal.

Tree Decomposition. For a tree decomposition (T, β) of a graph G, if T is a path, then (T, β) is also called a *path decomposition*, and if T is a caterpillar then (T, β) is also called a *caterpillar decomposition*. A *clique path* (*clique caterpillar*) of a graph G is a path decomposition (resp. *caterpillar decomposition*) of G where every bag is a distinct maximal clique. We remark that not every graph admits a clique caterpillar.

[4] Like other papers on this topic, we abuse the standard usage of the term AT in the literature, which refers to a triple of vertices such that each pair is joined by a path that avoids the neighborhood of the other vertex. Our usage and the standard one are "almost equivalent" (see, e.g., [10]).

Modules. Let G be a graph. A subset $M \subseteq V(G)$ is a *module* if for all $u, w \in M$ and $v \in V(G) \setminus M$, either both u and w are adjacent to v or both u and w are not adjacent to v. A module is *nontrivial* if neither $V(M) = \emptyset$ nor $V(M) = V(G)$.

A *modular tree decomposition* of a graph $G = (V, E)$ is a linear-size representation of all its modules. It consists of a rooted tree T, a function $f : V(T) \to 2^{V(G)}$ and a function $g : V(T) \to \{0, 1\}$, which in particular satisfy the following properties:

1. M is a module of G if and only if there is a node $v \in V(T)$ for which, either $M = f(v)$, or both $g(v) = 1$ and there is a subset U of the set of children of v such that $M = \bigcup_{u \in U} f(u)$.
2. Every $v, u \in V(T)$ that have the same parent in T satisfy $f(v) \cap f(u) = \emptyset$.
3. For every $v \in V(T)$, $\bigcup_{u \in \mathsf{child}(v)} f(u) = f(v)$.
4. $|V(T)| \leq 2n - 1$.

Furthermore, no node in T has exactly one child. Every graph G admits a modular tree decomposition, which can be constructed in $O(n^2)$ time and $O(n)$ space [47].

Hitting Chordless Cycles and Small Obstructions. We first state the following corollary of the results of Kim and Kwon [25].

▶ **Corollary 2.** *Let G be a graph, and let $k \in \mathbb{N}$. At least one of the following conditions holds:* **(i)** *G has k vertex-disjoint obstructions;* **(ii)** *there exists a subset $D \subseteq V(G)$ of size $\mathcal{O}(k^2 \log k)$ such that $G - D$ is a chordal graph that has no obstruction on at most $\max\{2k, 10\}$ vertices.*

3 The Case of a Clique Caterpillar

This section analyzes the Erdős-Pósa Property of ATs on graphs with a clique caterpillar. Let us begin with a definition.

▶ **Definition 3.** Let G be a graph. A clique caterpillar (T, β) of G is *nice* if every shallow vertex belongs to the bag of only one node of T and that node is a leaf.

The objective of this section is to prove the following lemma.

▶ **Lemma 4.** *Let $k \in \mathbb{N}$, and let G be a graph with a nice clique caterpillar (T, β), such that G is chordal and has no obstruction on at most ten vertices.*[5] *Then, at least one of the following conditions holds:* **(i)** *G has k vertex-disjoint ATs;* **(ii)** *there exists a subset $D \subseteq V(G)$ of size $\mathcal{O}(k^2)$ such that $G - D$ is an interval graph.*

To simplify statements in this section, let us fix $k \in \mathbb{N}$ and a chordal graph G with a nice clique caterpillar (T, β), which has no obstruction on at most ten vertices. Thus, whenever we discuss an obstruction in G, that obstruction is necessarily an AT on more than ten vertices. Moreover, let us fix a central path of T, and call it P. We denote $P = p_1 - p_2 - \cdots - p_d$ for $d = |V(P)|$. We think of P as a path oriented from p_1 to p_d. For a vertex $v \in V(G)$, we let $\mathsf{first}(v)$ be the first node p on P such that $v \in \beta(p)$ (if such a vertex does not exist, define $\mathsf{first}(v) = \mathsf{nil}$), and we let $\mathsf{last}(v)$ be the last node p on P such that $v \in \beta(p)$ (if such a vertex does not exist, define $\mathsf{last}(v) = \mathsf{nil}$). The notation $p_i < p_j$ means that $i < j$ (similarly, we define $\leq$). Note that as non-terminal vertices of an AT have non-adjacent neighbors, we have the following observation.

[5] We remark that the existence of the clique caterpillar already implies that G is chordal [18, 8].

▶ **Observation 5.** *Let $\mathbb{O}$ be an AT in G. For every non-terminal vertex v of $\mathbb{O}$, there exists $p \in V(P)$ such that $v \in \beta(p)$.*

Observation 5 implies that the notation presented next is well defined. In what follows, when we consider an AT $\mathbb{O}$, we index the base vertices $b_1^{\mathbb{O}}, b_2^{\mathbb{O}}, \ldots, b_{\eta^{\mathbb{O}}}^{\mathbb{O}}$ such that $\mathsf{first}(b_1^{\mathbb{O}}) \leq \mathsf{first}(b_{\eta^{\mathbb{O}}}^{\mathbb{O}})$. When $\mathbb{O}$ is clear from context, we simplify the notation, also in the context of terminal and center vertices.[6] Note that $\eta \geq 5$, as G does not have ATs on at most ten vertices (we use this observation implicitly throughout, e.g. to assume that $b_1, b_2, b_{\eta-2}, b_{\eta-1}$ and b_η are distinct vertices). We remark that clearly, for all $i \in \{2, 3, \ldots, \eta - 1\}$, $\mathsf{first}(b_i) \leq \mathsf{last}(b_{i-1}) < \mathsf{first}(b_{i+1})$ (also stated as Proposition 8.4 in [11]).

Our analysis relies on a notion of a special type of obstruction, defined by Cao and Marx [11], to exploit the "almost linear nature" of a caterpillar. To this end, we have the following notation. Given an AT $\mathbb{O}$, $\widehat{N}(\mathbb{O})$ denotes the set of vertices $v \in V(G)$ such that v is adjacent to every vertex in $\mathsf{base}(\mathbb{O})$. We also need to give three definitions.

▶ **Definition 6** ([11])**.**

(i) An AT $\mathbb{O}$ in G is *minimal* if there does not exist an AT $\mathbb{O}'$ such that $\mathsf{last}(b_1) \leq \mathsf{last}(b'_1) \leq \mathsf{first}(b'_{\eta'}) \leq \mathsf{first}(b_\eta)$, and $\mathsf{last}(b_1) < \mathsf{last}(b'_1)$ or $\mathsf{first}(b'_{\eta'}) < \mathsf{first}(b_\eta)$.

(ii) An AT $\mathbb{O}$ in G is *short* if $P(\mathbb{O})$ is a shortest path between t_ℓ and t_r in $G[\beta(p_i) \cup \beta(p_{i+1}) \cup \ldots \cup \beta(p_j) \cup \{t_\ell, t_r\}] - \widehat{N}(\mathsf{base}(\mathbb{O}))$, where $p_i = \mathsf{last}(b_1)$ and $p_j = \mathsf{first}(b_\eta)$.

(iii) An AT $\mathbb{O}$ in G is *first* if there does not exist an AT $\mathbb{O}'$ such that $\mathsf{first}(b'_{\eta'}) < \mathsf{first}(b_\eta)$.

We say that an AT is *good* if it is first, minimal and short. The following proposition asserts that a good AT exists. In this context, recall that we implicitly assume that G is not an arbitrary graph, but in particular it is a graph that has a nice clique caterpillar.

▶ **Proposition 7** (Lemma 8.8 & Proof of Theorem 2.4 (Page 31) [11])**.** *If G is not an interval graph, then it has a good AT.*

▶ **Proposition 8** (Claim 5 [11])**.** *Let $\mathbb{O}$ be a good AT. For any vertex $v \in (\beta(p_1) \cup \beta(p_2) \cup \ldots \beta(p_i)) \setminus \widehat{N}(\mathbb{O})$, where $p_i = \mathsf{first}(b_{\eta-2})$, it holds that v is not adjacent to any vertex that is shallow in G.*

Procedure `SeparateProcedure`. Let us consider the following procedure, which we call `SeparateProcedure`. Initialize $G^1 = G$ and $i = 1$. Now, as long as G^i is not an interval graph, we execute the following procedure:

1. Let $\mathbb{O}^i$ be a good AT in G^i, whose existence is guaranteed by Proposition 7.
2. Denote $p_j = \mathsf{first}(b^i_{\eta^i-2})$ and $p_q = \mathsf{last}(b^i_1)$. For all $\delta \in [d]$, denote $\beta^i(p_\delta) = \beta(p_\delta) \cap V(G^i)$. Let $\gamma^i = \gamma$ be the index in $\{q, q+1, \ldots, j-1\}$ such that,
   - there does not exist an index $\delta \in \{\gamma+1, \gamma+2, \ldots, j-1\}$ such that $|(\beta^i(p_\delta) \cap \beta^i(p_{\delta+1})) \setminus \widehat{N}(\mathbb{O}^i)| < 8k$, and
   - $|(\beta^i(p_\gamma) \cap \beta^i(p_{\gamma+1})) \setminus \widehat{N}(\mathbb{O}^i)| < 8k$.

   If such an index γ does not exist, define $\gamma = \mathsf{nil}$. Intuitively, γ is the largest index of a "small" separator in $G^i \setminus \widehat{N}(\mathbb{O})$ between b^i_1 and $b^i_{\eta-2}$.
3. Denote $S^i = (\beta^i(p_\gamma) \cap \beta^i(p_{\gamma+1})) \setminus \widehat{N}(\mathbb{O}^i)$ if $\gamma \neq \mathsf{nil}$, and $S^i = \emptyset$ otherwise.
4. Define $G^{i+1} = G^i - ((V(\mathbb{O}^i) \setminus \mathsf{base}(\mathbb{O}^i)) \cup \{b^i_1, b^i_2, b^i_3, b^i_{\eta-3}, b^i_{\eta-2}, b^i_{\eta-1}, b^i_{\eta^i}\} \cup S^i)$.
5. Increment i by 1.

[6] For example, if we consider an AT denoted by $\mathbb{O}$, $\mathbb{O}'$ and $\mathbb{O}^i$, then we use b_1 (b_η), b'_1 ($b'_{\eta'}$), b^i_1 ($b^i_{\eta^i}$) to refer to the first (last) base vertex of $\mathbb{O}$, $\mathbb{O}'$ and $\mathbb{O}^i$, respectively.

Let $i^\star$ denote the last index i considered by `SeparateProcedure`. In particular, $G^{i^\star}$ is an interval graph. Let us denote $S^\star = V(G) \setminus V(G^{i^\star})$. Then, $G - S^\star$ is an interval graph. Furthermore, note that $|S^\star| = \mathcal{O}(i^\star \cdot k)$.

▶ **Observation 9.** *If $i^\star \leq 2k$, then $S^\star \subseteq V(G)$ is a set of size $\mathcal{O}(k^2)$ such that $G - S^\star$ is an interval graph.*

Thus, to prove Lemma 4, it is sufficient to prove the following claim.

▶ **Lemma 10.** *If $i^\star > 2k$, then G has k vertex-disjoint obstructions.*

In what follows, we suppose that $i^\star > 2k$. To prove this lemma, we first need to introduce the following definitions.

▶ **Definition 11.** Let $i \in [2k]$. We say that an AT $\mathbb{O}$ in G is *i-relevant* if it is an AT in G^i, $t = t^i$, $t_\ell = t^i_\ell$, $t_r = t^i_r$, $c_1 = c^i_1$, $c_2 = c^i_2$, $b_1 = b^i_1$, $b_2 = b^i_2$, $b_{\eta-2} = b^i_{\eta^i-2}$, $b_{\eta-1} = b^i_{\eta^i-1}$ and $b_\eta = b^i_{\eta^i}$.[7] If in addition $b_3 = b^i_3$ and $b_{\eta-3} = b^i_{\eta-3}$, then we say that $\mathbb{O}$ is *highly i-relevant*.

▶ **Definition 12.** A tuple $(\widehat{\mathbb{O}}^1, \widehat{\mathbb{O}}^2, \ldots, \widehat{\mathbb{O}}^{2k})$ is *relevant* if for all $i \in [2k]$, $\widehat{\mathbb{O}}^i$ is i-relevant.

We further need the following notation. For every $i \in [2k]$, $\mathsf{before}(i) = \beta(p_1) \cup \beta(p_2) \cup \ldots \cup \beta(p_{\gamma^i})$ if $\gamma^i \neq \mathsf{nil}$ and $\mathsf{before}(i) = \emptyset$ otherwise. The heart of the proof of Lemma 10 is given by two statements. Towards the first one, let us first prove the following claim.

▶ **Lemma 13 (*).** *For all $i, i' \in [2k]$ where $i > i'$, i-relevant AT $\mathbb{O}$ and i'-relevant AT $\mathbb{O}'$, it holds that,* (1) *$V(\mathbb{O}) \cap V(\mathbb{O}') \cap \mathsf{before}(i') \subseteq \{t_\ell, b_1\}$;* (2) *$|V(\mathbb{O}) \cap V(\mathbb{O}') \cap \mathsf{before}(i')| \leq 1$; and* (3) *if $b_1 \in V(\mathbb{O}) \cap V(\mathbb{O}') \cap \mathsf{before}(i')$ then $t_\ell \notin (\bigcup_{i \in [d]} \beta(p_i))$.*

We now present the first statement that lies at the heart of the proof.

▶ **Lemma 14 (*).** *For all $i, i', \widehat{i} \in [2k]$ such that $i > i' > \widehat{i}$, i-relevant AT $\mathbb{O}$, i'-relevant AT $\mathbb{O}'$ and $\widehat{i}$-relevant AT $\widehat{\mathbb{O}}$, for at least one index $j \in \{i', \widehat{i}\}$, the following condition holds: $V(\mathbb{O}) \cap V(\mathbb{O}^j) \cap \mathsf{before}(j) = \emptyset$.*

▶ **Corollary 15 (*).** *Let $(\widehat{\mathbb{O}}^1, \widehat{\mathbb{O}}^2, \ldots, \widehat{\mathbb{O}}^{2k})$ be a relevant tuple. There exist k indices, $i_1 < i_2 < \ldots < i_k$, so that for every two indices $x, y \in \{i_1, i_2, \ldots, i_k\}$ where $x < y$, $V(\widehat{\mathbb{O}}^y) \cap V(\widehat{\mathbb{O}}^x) \cap \mathsf{before}(x) = \emptyset$.*

Towards the statement of the second lemma that lies at the heart of our proof, let us first state an immediate observation and one additional lemma.

▶ **Observation 16.** *An AT in G can contain at most four vertices of a clique in G.*

▶ **Lemma 17 (*).** *Let $(\widehat{\mathbb{O}}^1, \widehat{\mathbb{O}}^2, \ldots, \widehat{\mathbb{O}}^{2k})$ be a relevant tuple. For all $i \in [2k-1]$, there exists a path in $G^i - \widehat{N}(\mathbb{O}^i)$ from a vertex in $S^i \cup \{b^i_1\}$ to $b^i_{\eta-2}$ that does not contain any of the vertices of the ATs $\widehat{\mathbb{O}}^{i+1}, \widehat{\mathbb{O}}^{i+2}, \ldots, \widehat{\mathbb{O}}^{2k}$.*

We are now ready to prove the second statement central to the proof of Lemma 10.

▶ **Lemma 18 (*).** *Let $(\widehat{\mathbb{O}}^1, \widehat{\mathbb{O}}^2, \ldots, \widehat{\mathbb{O}}^{2k})$ be a relevant tuple. For all $i \in [2k-1]$ such that $\mathbb{O}^i$ is a highly i-relevant good AT in G^i, there exists an i-relevant AT $\mathbb{O}'$ such that the following condition holds: the base path of $\mathbb{O}'$ has a subpath Q from a vertex in $S^i \cup \{b^i_1\}$ to b^i_η that does not contain any of the vertices of the ATs $\widehat{\mathbb{O}}^{i+1}, \widehat{\mathbb{O}}^{i+2}, \ldots, \widehat{\mathbb{O}}^{2k}$.*

[7] That is, $\mathbb{O}$ and the AT $\mathbb{O}^i$ considered in the i-th iteration of `SeparateProcedure` have the same terminals, centers and two first and three last base vertices.

▶ **Corollary 19 (*).** *There exists a relevant tuple* $(\widehat{\mathbb{O}}^1, \widehat{\mathbb{O}}^2, \ldots, \widehat{\mathbb{O}}^{2k})$ *such that for all* $i \in [2k]$*, the following condition holds: the base path of* $\widehat{\mathbb{O}}^i$ *has a subpath* Q *from a vertex in* $S^i \cup \{b_1^i\}$ *to* b_η^i *that does not contain any of the vertices of the ATs* $\widehat{\mathbb{O}}^{i+1}, \widehat{\mathbb{O}}^{i+2}, \ldots, \widehat{\mathbb{O}}^{2k}$*.*

We are now ready to prove Lemma 10.

Proof of Lemma 10. By Corollary 19, there exists a relevant tuple $(\widehat{\mathbb{O}}^1, \widehat{\mathbb{O}}^2, \ldots, \widehat{\mathbb{O}}^{2k})$ such that for all $i \in [2k]$, the following condition holds: the base path of $\widehat{\mathbb{O}}^i$ has a subpath Q from a vertex in $S^i \cup \{b_1^i\}$ to b_η^i that does not contain any of the vertices of the ATs $\widehat{\mathbb{O}}^{i+1}, \widehat{\mathbb{O}}^{i+2}, \ldots, \widehat{\mathbb{O}}^{2k}$. By Corollary 15, there exist k indices, $i_1 < i_2 < \ldots < i_k$, such that for every two indices $x, y \in \{i_1, i_2, \ldots, i_k\}$ where $x < y$, $V(\widehat{\mathbb{O}}^y) \cap V(\widehat{\mathbb{O}}^x) \cap \mathsf{before}(x) = \emptyset$. Without loss of generality, suppose that $i_1 = 1, i_2 = 2, \ldots, i_k = k$ (the arguments to follow hold for any $i_1 < i_2 < \ldots < i_k$).

We claim that $\widehat{\mathbb{O}}^1, \widehat{\mathbb{O}}^2, \ldots, \widehat{\mathbb{O}}^k$ are vertex disjoint, which would complete the proof. To prove this claim, we arbitrarily choose $i, j \in [k]$ such that $i < j$. First note that as $\widehat{\mathbb{O}}^i$ and $\widehat{\mathbb{O}}^j$ are i-relevant and j-relevant, we have that the terminals and centers of $\widehat{\mathbb{O}}^i$ do not belong to $\widehat{\mathbb{O}}^j$. Moreover, the base path of $\widehat{\mathbb{O}}^i$ has a subpath Q from a vertex $v^\star$ in $S^i \cup \{b_1^i\}$ to $b_{\eta^i}^i$ that has no vertex of $\widehat{\mathbb{O}}^j$. Let W denote the subpath of the base path of $\widehat{\mathbb{O}}^i$ from b_1^i to $v^\star$. Hence, to conclude that $\widehat{\mathbb{O}}^i$ and $\widehat{\mathbb{O}}^j$ are vertex disjoint, it remains to show that no vertex of W belongs to $\widehat{\mathbb{O}}^j$. Notice that $V(W) \subseteq V(\widehat{\mathbb{O}}^i) \cap \mathsf{before}(i)$. By our choice of $(\widehat{\mathbb{O}}^1, \widehat{\mathbb{O}}^2, \ldots, \widehat{\mathbb{O}}^k)$, it holds that $V(\widehat{\mathbb{O}}^i) \cap \mathsf{before}(i)$ does not have any vertex of $\widehat{\mathbb{O}}^j$. Thus, the proof is complete. ◀

4 Decomposition of Modules

Let us begin with the following simple observation, on which we rely implicitly in our arguments, and which follows immediately from the definition of a modular tree decomposition. For simplicity, we use the abbreviations $f|_v = f|_{V(T|_v)}$ and $g|_v = g|_{V(T|_v)}$.

▶ **Observation 20.** *Let* G *be a graph with a modular tree decomposition* (T, f, g)*, and let* $v \in V(T)$*. Then,* $(T|_v, f|_v, g|_v)$ *is a modular tree decomposition of* $G[f(v)]$*.*

We proceed by introducing the definition of a *problematic set* and a *problematic node*.

▶ **Definition 21.** Let G be a graph with a modular tree decomposition (T, f, g). The *set of problematic obstructions* of a node $v \in V(T)$, denoted by $\mathsf{prob}_G(v)$, is the set of all obstructions $\mathbb{O}$ in $G[f(v)]$ such that for every child u of v in T, $\mathbb{O}$ is not an obstruction in $G[f(u)]$, that is, $V(\mathbb{O}) \setminus f(u) \neq \emptyset$. When G is clear from context, it is omitted.

▶ **Definition 22.** Let G be a graph with a modular tree decomposition (T, f, g). A node $v \in V(T)$ is *problematic* if $\mathsf{prob}(v) \neq \emptyset$. The set of problematic nodes is denoted by $\mathsf{prob}_G(T)$. When G is clear from context, it is omitted.

▶ **Observation 23.** *Let* G *be a graph with a modular tree decomposition* (T, f, g)*. The sets* $\mathsf{prob}(v)$*,* $v \in \mathsf{prob}(T)$*, define a partition of the set of obstructions of* G*. That is, for all* $u, v \in V(T)$*,* $\mathsf{prob}(u) \cap \mathsf{prob}(v) = \emptyset$*, and the set of obstructions of* G *is precisely* $\bigcup_{v \in \mathsf{prob}(T)} \mathsf{prob}(v)$*.*

We argue that nodes assigned 1 by g are non-problematic. And further, a problematic node should have "many" children.

▶ **Lemma 24 (*).** *Let* G *be a graph that has no obstruction on at most* $\max\{2k, 10\}$ *vertices. Let* (T, f, g) *be a modular tree decomposition of* G*, and let* $v \in V(T)$ *such that* $g(v) = 1$*. Then,* v *is not a problematic node.*

▶ **Lemma 25** (*). *Let G be a graph that has no obstruction on at most $\max\{2k, 10\}$ vertices. Let (T, f, g) be a modular tree decomposition of G, and let $v \in V(T)$ be a problematic node. Then, v has at least $\max\{2k, 10\} + 1$ children in T.*

In order to proceed, we need the following definition and notation.

▶ **Definition 26.** Let G be a graph with a modular tree decomposition (T, f, g). A subset $P \subseteq \mathsf{prob}(T)$ has a conflict if there exist $u, v \in P$ such that v is a descendant of u in T and on the (unique) path between u and v in T no vertex belongs to $\mathsf{prob}(T) \setminus P$.

▶ **Definition 27.** Let G be a graph with a modular tree decomposition (T, f, g). For a node $v \in V(T)$, $\mathsf{pack}_G(v)$ is the maximum number of vertex-disjoint obstructions in $\mathsf{prob}(v)$. When G is clear from context, it is omitted.

Note that a problematic node is precisely a node such that $\mathsf{pack}(v) \geq 1$.

▶ **Lemma 28** (*). *Let G be a graph that has no obstruction on at most $\max\{2k, 10\}$ vertices, and which does not have k vertex-disjoint obstructions. Let (T, f, g) be a modular tree decomposition of G. Let $P \subseteq \mathsf{prob}(T)$ with no conflicts. Then, for each $v \in P$ and each child u of v in T such that u has a problematic descendant, there exist at least k vertices in $f(u)$ that do not belong to $\bigcup_w f(w)$ where w ranges over all nodes in P that are descendants of u in T.*

▶ **Lemma 29** (*). *Let G be a graph that has no obstruction on at most $\max\{2k, 10\}$ vertices. Let (T, f, g) be a modular tree decomposition of G. Let $P \subseteq \mathsf{prob}(T)$ with no conflicts. Then, G has $\min\{k, \sum_{v \in P} \mathsf{pack}(v)\}$ vertex-disjoint obstructions.*

We also show that $\mathsf{prob}(T)$ can be divided into two sets with no conflicts.

▶ **Lemma 30** (*). *Let G be a graph with a modular tree decomposition (T, f, g). There exists a partition (P_1, P_2) of $\mathsf{prob}(T)$ such that neither P_1 has a conflict nor P_2 has a conflict.*

Specific classes of graphs, called *reduced graphs* and *nice interval graphs*, were defined by Cao and Marx as follows.

▶ **Definition 31** ([11]). A graph G is *reduced* if it satisfies the following properties: (i) Every non-trivial module of G is a clique, and (ii) G does not have any obstruction on at most ten vertices.

▶ **Definition 32** ([11]). A graph G is *nice* if it satisfies the following properties: (i) G is chordal; (ii) G does not have any obstruction on at most ten vertices; and (iii) every vertex in G that is a shallow terminal of at least one obstruction is simplicial.

These definitions were in particular used to derive the following results.

▶ **Proposition 33** (Theorem 2.1 [11]). *Let G be a reduced graph. Every vertex in G that is a shallow terminal of at least one obstruction is simplicial.*

▶ **Proposition 34** (Proposition 8.3 [11]). *Any nice graph has a nice clique caterpillar (T, β).*

▶ **Corollary 35.** *Any chordal reduced graph has a nice clique caterpillar (T, β).*

Let us derive a consequence of Corollary 35 with respect to a modular tree decomposition.

▶ **Lemma 36** (*). *Let G be a chordal graph that has no obstruction on at most $\max\{2k, 10\}$ vertices. Let (T, f, g) be a modular tree decomposition of G, and let $v \in V(T)$ be a problematic node such that for every child u of v in T, $G[f(u)]$ is a clique. Then, $G[f(v)]$ has a nice clique caterpillar.*

Towards the proof of the main result of this section, we need one additional notation.

▶ **Definition 37.** Let G be a graph with a modular tree decomposition (T, f, g), and let $v \in V(T)$. Then, $\mathsf{clique}(G, v)$ denotes the graph obtained from G by turning each $G[f(u)]$, $u \in \mathsf{child}(v)$, into a clique. That is, $V(\mathsf{clique}(G, v)) = V(G)$ and $E(\mathsf{clique}(G, v)) = E(G) \cup (\bigcup_{u \in \mathsf{child}(v)} \{\{x, y\} : x, y \in f(u)\})$.

▶ **Lemma 38 (*).** *Let G be a graph that has no obstruction on at most $\max\{2k, 10\}$ vertices. Let (T, f, g) be a modular tree decomposition of G, and let $v \in V(T)$. Then, the set of obstructions in $\mathsf{clique}(G, v)[f(v)]$ is precisely $\mathsf{prob}_G(v)$.*

▶ **Lemma 39 (*).** *Let $k \in \mathbb{N}$, and let G be a chordal graph that has no obstruction on at most $\max\{2k, 10\}$ vertices. Let (T, f, g) be a modular tree decomposition of G, and let $v \in V(T)$. Then, at least one of the following conditions holds:* **(i)** *$\mathsf{pack}(v) \geq k$;* **(ii)** *there exists a subset $D \subseteq V(G)$ of size $\mathcal{O}(k^2)$ that intersects the vertex set of every obstruction in $\mathsf{prob}(v)$.*

We are now ready to prove the main result of this section.

▶ **Lemma 40.** *Let $k \in \mathbb{N}$, and let G be a chordal graph that has no obstruction on at most $\max\{2k, 10\}$ vertices. Then, at least one of the following conditions holds:* **(i)** *G has k vertex-disjoint obstructions;* **(ii)** *there exists a subset $D \subseteq V(G)$ of size $\mathcal{O}(k^2)$ such that $G - D$ is an interval graph.*

Proof. Suppose that G does not have k vertex-disjoint obstructions, else we are done. By Lemma 30, there exists a partition (P_1, P_2) of $\mathsf{prob}(T)$ such that neither P_1 has a conflict nor P_2 has a conflict. By Lemma 29, for each $i \in [2]$, G has $\sum_{v \in P_i} \mathsf{pack}(v)$ vertex-disjoint obstructions. Thus, by Observation 23, for each $i \in [2]$, $|\sum_{v \in P_i} \mathsf{pack}(v)| < k$. This means that $\sum_{v \in \mathsf{prob}(T)} \mathsf{pack}(v) < 2k$.

By Lemma 39, for all $v \in \mathsf{prob}(T)$, there exists a subset $D_v \subseteq V(G)$ of size $\mathcal{O}((\mathsf{pack}(v)+1)^2)$ that intersects the vertex set of every obstruction in $\mathsf{prob}(v)$. Denote $D = \bigcup_{v \in \mathsf{prob}(T)} D_v$. Then, $|D| = \mathcal{O}(\sum_{v \in \mathsf{prob}(T)} (\mathsf{pack}(v) + 1)^2)$. By Observation 23, we have that $G - D$ is an interval graph. Thus, to conclude the proof, it remains to show that $\sum_{v \in \mathsf{prob}(T)} (\mathsf{pack}(v) + 1)^2 = \mathcal{O}(k^2)$. Since for all $v \in \mathsf{prob}(T)$, $\mathsf{pack}(v) \geq 1$, it is sufficient to show that $\sum_{v \in \mathsf{prob}(T)} (\mathsf{pack}(v))^2 = \mathcal{O}(k^2)$. Recall that $\sum_{v \in \mathsf{prob}(T)} \mathsf{pack}(v) < 2k$. Thus, $\sum_{v \in \mathsf{prob}(T)} (\mathsf{pack}(v))^2 \leq (\sum_{v \in \mathsf{prob}(T)} \mathsf{pack}(v)) \cdot (\sum_{v \in \mathsf{prob}(T)} \mathsf{pack}(v)) < 2k \cdot 2k = \mathcal{O}(k^2)$. This completes the proof. ◀

5 Putting It All Together

Finally, we are ready to prove our main theorem.

Proof of Theorem 1. By Corollary 2, at least one of the following conditions hold: **(i)** G has k vertex-disjoint obstructions; **(ii)** there exists a subset $D' \subseteq V(G)$ of size $\mathcal{O}(k^2 \log k)$ such that $G - D'$ is a chordal graph that has no obstruction on at most $\max\{2k, 10\}$ vertices. In the first case, our proof is complete, and thus we next suppose that the second case applies. Then, by Lemma 40, at least one of the following conditions hold: **(i)** $G - D'$ has k vertex-disjoint obstructions; **(ii)** there exists a subset $\widehat{D} \subseteq V(G)$ of size $\mathcal{O}(k^2)$ such that $(G - D') - \widehat{D}$ is an interval graph. In the first case, our proof is complete. In the second case, we have that $D = D' \cup \widehat{D}$ is a set of size $\mathcal{O}(k^2 \log k)$ such that $G - D$ is an interval graph, which again completes the proof. ◀

Before we turn to prove a corollary of our main theorem, we need one more proposition.

▶ **Proposition 41** ([10]). *There exists an $\mathcal{O}(nm)$-time algorithm that, given a graph G, outputs an integer d' such that the following conditions hold:* **(i)** *there exists a subset $D' \subseteq V(G)$ of size at most d' such that $G - D'$ is an interval graph;* **(ii)** *$d' \leq 8d$ for the integer d that is the minimum size of a subset $D \subseteq V(G)$ such that $G - D$ is an interval graph.*

As a consequence of Theorem 1 and Proposition 41, we derive the following corollary.

▶ **Corollary 42** (*). *There exist a constant $c \in \mathbb{N}$ and an $\mathcal{O}(nm)$-time algorithm that, given a graph G and an integer $k \in \mathbb{N}$, correctly concludes which one of the following conditions holds:*[8] **(i)** *G has k vertex-disjoint obstructions;* **(ii)** *there exists a subset $D \subseteq V(G)$ of size $ck^2 \log k$ such that $G - D$ is an interval graph.*

References

1 Noga Alon. Covering a hypergraph of subgraphs. *Discrete Mathematics*, 257(2-3):249–254, 2002.

2 Saeed Akhoondian Amiri, Ken-ichi Kawarabayashi, Stephan Kreutzer, and Paul Wollan. The erdos-posa property for directed graphs. *CoRR*, abs/1603.02504, 2016. `arXiv:1603.02504`.

3 Julio Aracena, Jacques Demongeot, and Eric Goles. Positive and negative circuits in discrete neural networks. *IEEE Transactions on Neural Networks*, 15(1):77–83, 2004.

4 Etienne Birmelé, J Adrian Bondy, and Bruce A Reed. The Erdős–Pósa property for long circuits. *Combinatorica*, 27(2):135–145, 2007.

5 John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*, volume 290. Citeseer, 1976.

6 Nicolas Bousquet. *Hitting sets: VC-dimension and Multicut.* PhD thesis, Université Montpellier II-Sciences et Techniques du Languedoc, 2013.

7 Nicolas Bousquet and Stéphan Thomassé. Vc-dimension and Erdős–Pósa property. *Discrete Mathematics*, 338(12):2302–2317, 2015.

8 Andreas Brandstädt, Van Bang Le, and Jeremy P Spinrad. *Graph classes: a survey.* SIAM, 1999.

9 Henning Bruhn, Felix Joos, and Oliver Schaudt. Long cycles through prescribed vertices have the Erdős–Pósa property. *Journal of Graph Theory*, 2017.

10 Yixin Cao. Linear recognition of almost interval graphs. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1096–1115. SIAM, 2016. `doi:10.1137/1.9781611974331.ch77`.

11 Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *ACM Trans. Algorithms*, 11(3):21:1–21:35, 2015. `doi:10.1145/2629595`.

12 Chandra Chekuri and Julia Chuzhoy. Large-treewidth graph decompositions and applications. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC)*, pages 291–300, 2013.

13 Reinhard Diestel. *Graph Theory.* Springer-Verlag, Heidelberg, 4th edition, 2010.

14 P Erdős and L Pósa. On independent circuits contained in a graph. *Canad. J. Math.*, 17:347–352, 1965.

15 Samuel Fiorini, Nadia Hardy, Bruce Reed, and Adrian Vetta. Approximate min–max relations for odd cycles in planar graphs. *Mathematical programming*, 110(1):71–91, 2007.

[8] In particular, at least one condition holds, and if both conditions hold, then the algorithm can choose either of the two conditions.

16 Jim Geelen and Kasper Kabell. The Erdős–Pósa property for matroid circuits. *Journal of Combinatorial Theory, Series B*, 99(2):407–419, 2009.

17 Archontia C. Giannopoulou, O-joung Kwon, Jean-Florent Raymond, and Dimitrios M. Thilikos. Packing and covering immersion-expansions of planar sub-cubic graphs. *Eur. J. Comb.*, 65:154–167, 2017. doi:10.1016/j.ejc.2017.05.009.

18 Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

19 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Science & Business Media, 2012.

20 Bertrand Guenin and Robin Thomas. Packing directed circuits exactly. *Combinatorica*, 31(4):397–421, 2011.

21 András Gyárfás and Jenö Lehel. A helly-type problem in trees. In *Colloquia mathematica Societatis Jànos Bolyai*, volume 4, pages 571–584, 1970.

22 Frédéric Havet and Ana Karolinna Maia. On disjoint directed cycles with prescribed minimum lengths. *Research Report RR-8286, INRIA*, 2013.

23 Winfried Hochstättler, Robert Nickel, and Britta Peis. Two disjoint negative cycles in a signed graph. *Electronic Notes in Discrete Mathematics*, 25:107–111, 2006.

24 Felix Joos. Parity linkage and the erdős-pósa property of odd cycles through prescribed vertices in highly connected graphs. *Journal of Graph Theory*, 85(4):747–758, 2017. doi:10.1002/jgt.22103.

25 O joung Kwon and Eun Jung Kim. Erdős–Pósa property of chordless cycles and its application. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, To Appear*, 2018.

26 Naonori Kakimura and Ken-ichi Kawarabayashi. Fixed-parameter tractability for subset feedback set problems with parity constraints. *Theoretical Computer Science*, 576:61–76, 2015.

27 Naonori Kakimura, Ken-ichi Kawarabayashi, and Dániel Marx. Packing cycles through prescribed vertices. *Journal of Combinatorial Theory, Series B*, 101(5):378–381, 2011.

28 Ken-ichi Kawarabayashi and Bruce Reed. Highly parity linked graphs. *Combinatorica*, 29(2):215–225, 2009.

29 Ken-ichi Kawarabayashi and Paul Wollan. Non-zero disjoint cycles in highly connected group labelled graphs. *Journal of Combinatorial Theory, Series B*, 96(2):296–301, 2006.

30 Dénes König. Über graphen und ihre anwendung auf determinantentheorie und mengenlehre. *Mathematische Annalen*, 77(4):453–465, 1916.

31 Chun-Hung Liu. Packing and covering immersions in 4-edge-connected graphs. *CoRR*, abs/1505.00867, 2015. URL: https://arxiv.org/abs/1505.00867.

32 Claudio L Lucchesi and DH Younger. A minimax theorem for directed graphs. *Journal of the London Mathematical Society*, 2(3):369–374, 1978.

33 Karl Menger. Zur allgemeinen kurventheorie. *Fund. Math.*, 10(4):96–115, 1927.

34 Frank Mousset, Andreas Noever, Nemanja Škorić, and Felix Weissenberger. A tight Erdős–Pósa function for long cycles. *Journal of Combinatorial Theory, Series B*, 125:21–32, 2017.

35 Matteo Pontecorvi and Paul Wollan. Disjoint cycles intersecting a set of vertices. *Journal of Combinatorial Theory, Series B*, 102(5):1134–1141, 2012.

36 Dieter Rautenbach and Bruce Reed. The Erdős–Pósa property for odd cycles in highly connected graphs. *Combinatorica*, 21(2):267–278, 2001.

37 Jean-Florent Raymond and Dimitrios M Thilikos. Recent techniques and results on the Erdős–Pósa property. *Discrete Applied Mathematics*, 2017.

38 Bruce Reed. *Tree Width and Tangles: A New Connectivity Measure and Some Applications*, page 87–162. Cambridge University Press, 1997.

39 Bruce Reed. Mangoes and blueberries. *Combinatorica*, 19(2):267–296, 1999.

40 Bruce Reed, Neil Robertson, Paul Seymour, and Robin Thomas. Packing directed circuits. *Combinatorica*, 16(4):535–554, 1996.

41 Neil Robertson, Paul D. Seymour, and Robin Thomas. Quickly excluding a planar graph. *J. Comb. Theory, Ser. B*, 62(2):323–348, 1994. doi:10.1006/jctb.1994.1073.

42 Neil Robertson and P.D Seymour. Graph minors. v. excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, 1986.

43 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2002.

44 Paul D. Seymour. Packing circuits in eulerian digraphs. *Combinatorica*, 16(2):223–231, 1996.

45 Miklós Simonovits. A new proof and generalizations of a theorem of Erdős–Pósa on graphs without k+ 1 independent circuits. *Acta Mathematica Hungarica*, 18(1-2):191–206, 1967.

46 Mechthild Stoer. *Design of survivable networks*. Springer, 2006.

47 Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645. Springer, 2008. doi:10.1007/978-3-540-70575-8_52.

48 Carsten Thomassen. The Erdős–Pósa property for odd cycles in graphs of large connectivity. *Combinatorica*, 21(2):321–333, 2001.

All Classical Adversary Methods are Equivalent for Total Functions

Andris Ambainis
Centre for Quantum Computer Science, Faculty of Computing, University of Latvia, Raiņa 19, Riga, Latvia, LV-1586
andris.ambainis@lu.lv

Martins Kokainis
Centre for Quantum Computer Science, Faculty of Computing, University of Latvia, Raiņa 19, Riga, Latvia, LV-1586
martins.kokainis,krisjanis.prusis@lu.lv

Krišjānis Prūsis
Centre for Quantum Computer Science, Faculty of Computing, University of Latvia, Raiņa 19, Riga, Latvia, LV-1586
krisjanis.prusis@lu.lv

Jevgēnijs Vihrovs
Centre for Quantum Computer Science, Faculty of Computing, University of Latvia, Raiņa 19, Riga, Latvia, LV-1586

Abstract

We show that all known classical adversary lower bounds on randomized query complexity are equivalent for total functions, and are equal to the fractional block sensitivity $\mathrm{fbs}(f)$. That includes the Kolmogorov complexity bound of Laplante and Magniez and the earlier relational adversary bound of Aaronson. For partial functions, we show unbounded separations between $\mathrm{fbs}(f)$ and other adversary bounds, as well as between the relational and Kolmogorov complexity bounds.

We also show that, for partial functions, fractional block sensitivity cannot give lower bounds larger than $\sqrt{n \cdot \mathrm{bs}(f)}$, where n is the number of variables and $\mathrm{bs}(f)$ is the block sensitivity. Then we exhibit a partial function f that matches this upper bound, $\mathrm{fbs}(f) = \Omega(\sqrt{n \cdot \mathrm{bs}(f)})$.

2012 ACM Subject Classification Theory of computation → Probabilistic computation

Keywords and phrases Randomized Query Complexity, Lower Bounds, Adversary Bounds, Fractional Block Sensitivity

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.8

Funding This work is supported by the ERC Advanced Grant MQC and Latvian State Research Programme NexIT Project No. 1.

Acknowledgements We are grateful to Rahul Jain for igniting our interest in the classical adversary bounds and Srijita Kundu and Swagato Sanyal for helpful discussions. We also thank Jānis Iraids for helpful discussions on block sensitivity versus fractional block sensitivity problem.

1 Introduction

Query complexity of functions is one of the simplest and most useful models of computation. It is used to show lower bounds on the amount of time required to solve a computational task, and to compare the capabilities of the quantum, randomized and deterministic models of

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 8; pp. 8:1–8:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

computation. Thus, providing lower bounds in the query model is essential in understanding the complexity of computational problems.

In the query model, an algorithm has to compute a function $f : S \to H$, given a string x from $S \subseteq G^n$, where G and H are finite alphabets. With a single query, it can provide the oracle with an index $i \in [n]$ and receive back the value x_i. After a number of queries (possibly, adaptive), the algorithm must compute $f(x)$. The cost of the computation is the number of queries made by the algorithm.

The query complexity of a function f in the deterministic setting is denoted by $\mathrm{D}(f)$ and is also called the decision tree complexity. The two-sided bounded-error randomized and quantum query complexities are denoted by $\mathrm{R}(f)$ and $\mathrm{Q}(f)$, respectively (which means that given any input, the algorithm must produce a correct answer with probability at least 2/3). For a comprehensive survey on the power of these models, see [9], and for the state-of-the-art relationships between them, see [3].

In this work, we investigate the relations among a certain set of lower bound techniques on $\mathrm{R}(f)$, called the classical adversary methods, and how they connect to other well-known lower bounds on the randomized query complexity.

1.1 Known Lower Bounds

One of the first general lower bound methods on randomized query complexity is Yao's minimax principle, which states that it is sufficient to exhibit a hard distribution on the inputs and lower bound the complexity of any deterministic algorithm under such distribution [21]. Yao's minimax principle is known to be optimal for any function but involves a hard-to-describe and hard-to-compute quantity (the complexity of the best deterministic algorithm under some distribution).

More concrete randomized lower bounds are block sensitivity $\mathrm{bs}(f)$ [16] and the approximate degree of the polynomial representing the function $\widetilde{\deg}(f)$ [17] introduced by Nisan and Szegedy. Afterwards, Aaronson extended the notion of the certificate complexity $\mathrm{C}(f)$ (a deterministic lower bound) to the randomized setting by introducing randomized certificate complexity $\mathrm{RC}(f)$ [2]. Following this result, both Tal and Gilmer, Saks and Srinivasan independently discovered the fractional block sensitivity $\mathrm{fbs}(f)$ lower bound [20, 10], which is equal to the fractional certificate complexity $\mathrm{FC}(f)$ measure, as respective dual linear programs. Since these measures are relaxations of block sensitivity and certificate complexity if written as integer programs, they satisfy the following hierarchy:

$$\mathrm{bs}(f) \leq \mathrm{fbs}(f) = \mathrm{FC}(f) \leq \mathrm{C}(f).$$

Perhaps surprisingly, fractional block sensitivity turned out to be equivalent to randomized certificate complexity, $\mathrm{fbs}(f) = \Theta(\mathrm{RC}(f))$. Approximate degree and fractional block sensitivity are incomparable in general, but it has been shown that $\mathrm{fbs}(f) \leq \widetilde{\deg}(f)^2$ [13] and $\widetilde{\deg}(f) \leq \mathrm{bs}(f)^3 \leq \mathrm{fbs}(f)^3$ [16, 7].

Currently one of the strongest lower bounds is the partition bound $\mathrm{prt}(f)$ of Jain and Klauck [12], which is larger than all of the above mentioned randomized lower bounds (even the approximate degree), and the classical adversary methods listed below. Its power is illustrated by the $\textsc{Tribes}_n$ function (an $\textsc{And}$ of $\sqrt{n}$ $\textsc{Or}$s on $\sqrt{n}$ variables), where it gives a tight $\Omega(n)$ lower bound, while all of the other lower bounds give only $O(\sqrt{n})$. Recently, Ben-David and Kothari introduced the randomized sabotage complexity measure $\mathrm{RS}(f)$ [8], which is an even stronger classical lower bound than the partition bound.

In a separate line of research, Ambainis gave a versatile quantum adversary lower bound method with a wide range of applications [4]. Since then, many generalizations of the

quantum adversary method have been introduced (see [19] for a list of known quantum adversary bounds). Several of these formulations have been lifted back to the randomized setting. Aaronson proved a classical analogue of Ambainis' relational adversary bound and used it to provide a lower bound for the local search problem [1]. Laplante and Magniez introduced the Kolmogorov complexity adversary bound for both quantum and classical settings and showed that it subsumes many other adversary techniques. [14]. They also gave a classical variation of Ambainis' adversary bound in a different way than Aaronson. Some of the other adversary methods like spectral adversary have not been generalized back to the randomized setting.

While some relations between the adversary bounds had been known before, Špalek and Szegedy proved that practically all known quantum adversary methods are in fact equivalent [19] (this excludes the general quantum adversary bound, which gives an exact estimate on quantum query complexity for all Boolean functions [11, 18]). This result cannot be immediately generalized to the classical setting, as the equivalence follows through the spectral adversary which has no classical analogue. They also showed that the quantum adversary cannot give lower bounds better than a certain "certificate complexity barrier". Recently, Kulkarni and Tal strengthened the barrier using fractional certificate complexity. Specifically, for any Boolean function f the quantum adversary is at most $\sqrt{\mathrm{FC}^0(f)\,\mathrm{FC}^1(f)}$, if f is total, and at most $2\sqrt{n \cdot \min\{\mathrm{FC}^0(f), \mathrm{FC}^1(f)\}}$, if f is partial [13].[1]

With the advances on the quantum adversary front, one could hope for a similar equivalence result to also hold for the classical adversary bounds. Some relations are known: Laplante and Magniez have shown that the Kolmogorov complexity lower bound is at least as strong as Aaronson's relational and Ambainis' weighted adversary bounds [14]. Jain and Klauck have noted that the minimax over probability distributions adversary bound is at most $\mathrm{C}(f)$ for total functions [12]. In general, the relationships among the classical adversary bounds until this point remained unclear.

1.2 Our Results

Our main result shows that the known classical adversary bounds are all equivalent for total functions. That includes Aaronson's relational adversary bound $\mathrm{CRA}(f)$, Ambainis' weighted adversary bound $\mathrm{CWA}(f)$, the Kolmogorov complexity adversary bound $\mathrm{CKA}(f)$ and the minimax over probability distributions adversary bound $\mathrm{CMM}(f)$. Surprisingly, they are equivalent to the fractional block sensitivity $\mathrm{fbs}(f)$.

We also add to this list a certain restricted version of the relational adversary bound. More specifically, we require that the relation matrix between the inputs has rank 1, and denote this (seemingly weaker) lower bound by $\mathrm{CRA}_1(f)$. Thus for total functions $\mathrm{CRA}(f) = \Theta\left(\mathrm{CRA}_1(f)\right)$, where the latter is much easier to calculate for Boolean functions.

All this shows that $\mathrm{fbs}(f)$ is a fundamental lower bound measure for total functions with many different formulations, including the previously known $\mathrm{FC}(f)$ and $\mathrm{RC}(f)$. Another interesting corollary is that since the quantum certificate complexity $\mathrm{QC}(f) = \Theta(\sqrt{\mathrm{RC}(f)})$ is a lower bound on the quantum query complexity [2], we have that by taking the square root of any of the adversary bounds above, we obtain a quantum lower bound for total functions.

Along the way, for partial functions we show the equivalence between $\mathrm{CRA}(f)$ and $\mathrm{CWA}(f)$, and also between $\mathrm{CKA}(f)$ and $\mathrm{CMM}(f)$. In the case of partial functions, $\mathrm{fbs}(f)$

[1] Here, $\mathrm{FC}^0(f)$ and $\mathrm{FC}^1(f)$ stand for the maximum fractional certificate complexity over negative and positive inputs, respectively.

becomes weaker than all these adversary methods. In particular, we show an example of a function where each of these adversary methods gives an $\Omega(n)$ lower bound, while fractional block sensitivity is $O(1)$. We also show that $\mathrm{CRA}(f)$ and $\mathrm{CMM}(f)$ are not equivalent for partial functions, as there exists an example where $\mathrm{CRA}(f)$ is constant, but $\mathrm{CMM}(f) = \Theta(\log n)$.

We also show a "block sensitivity" barrier for fractional block sensitivity. Namely, for any partial function f, the fractional block sensitivity is at most $\sqrt{n \cdot \mathrm{bs}(f)}$. Note that the adversary bounds do not bear this limitation, as witnessed by the aforementioned example. This result is tight, as we exhibit a partial function that matches this upper bound.

Even though our results are similar to the quantum case in [19] in spirit, the proof methods are different.

2 Preliminaries

In this section we define the complexity measures we are going to work with in the paper. In the following definitions and the rest of the paper consider f to be a partial function $f : S \to H$ with domain $S \subseteq G^n$, where G, H are some finite alphabets and n is the length of the input string. Throughout the paper we assume that f is not constant.

2.1 Block Sensitivity

For $x \in S$, a subset of indices $B \subseteq [n]$ is a *sensitive block* of x if there exists a y such that $f(x) \neq f(y)$ and $B = \{i \mid x_i \neq y_i\}$. The *block sensitivity* $\mathrm{bs}(f, x)$ of f on x is the maximum number k of disjoint subsets $B_1, \ldots, B_k \subseteq [n]$ such that B_i is a sensitive block of x for each $i \in [k]$. The block sensitivity of f is defined as $\mathrm{bs}(f) = \max_{x \in S} \mathrm{bs}(f, x)$.

Let $\mathcal{B} = \{B \mid \exists y : f(x) \neq f(y) \text{ and } B = \{i \mid x_i \neq y_i\}\}$ be the set of sensitive blocks of x. The *fractional block sensitivity* $\mathrm{fbs}(f, x)$ of f on x is defined as the optimal value of the following linear program:

$$\text{maximize} \sum_{B \in \mathcal{B}} w_x(B) \qquad \text{subject to} \quad \forall i \in [n] : \sum_{\substack{B \in \mathcal{B} \\ i \in B}} w_x(B) \leq 1.$$

Here, $w_x : \mathcal{B} \to [0; 1]$. The fractional block sensitivity of f is defined as $\mathrm{fbs}(f) = \max_{x \in S} \mathrm{fbs}(f, x)$.

When the weights are taken as either 0 or 1, the optimal solution to the corresponding integer program is equal to $\mathrm{bs}(f, x)$. Hence $\mathrm{fbs}(f, x)$ is a relaxation of $\mathrm{bs}(f, x)$, and we have $\mathrm{bs}(f, x) \leq \mathrm{fbs}(f, x)$.

2.2 Certificate Complexity

An *assignment* is a map $A : \{1, \ldots, n\} \to G \cup \{*\}$. Informally, the elements of G are the values fixed by the assignment and * is a wildcard symbol that can be any letter of G. A string $x \in S$ is said to be consistent with A if for all $i \in [n]$ such that $A(i) \neq *$, we have $x_i = A(i)$. The length of A is the number of positions that A fixes to a letter of G.

For an $h \in H$, an h-certificate for f is an assignment A such that for all strings $x \in A$ we have $f(x) = h$. The *certificate complexity* $\mathrm{C}(f, x)$ of f on x is the size of the shortest $f(x)$-certificate that x is consistent with. The certificate complexity of f is defined as $\mathrm{C}(f) = \max_{x \in S} \mathrm{C}(f, x)$.

The *fractional certificate complexity* $\mathrm{FC}(f,x)$ of f on $x \in S$ is defined as the optimal value of the following linear program:

$$\text{minimize} \sum_{i\in[n]} v_x(i) \qquad \text{subject to} \quad \forall y \in S \text{ s.t. } f(x)\neq f(y): \sum_{i:x_i\neq y_i} v_x(i) \geq 1.$$

Here, $v_x : [n] \to [0;1]$ for each $x \in S$. The fractional certificate complexity of f is defined as $\mathrm{FC}(f) = \max_{x\in S} \mathrm{FC}(f,x)$.

When the weights are taken as either 0 or 1, the optimal solution to the corresponding integer program is equal to $\mathrm{C}(f,x)$. Hence $\mathrm{FC}(f,x)$ is a relaxation of $\mathrm{C}(f,x)$, and we have $\mathrm{FC}(f,x) \leq \mathrm{C}(f,x)$.

It has been shown that $\mathrm{fbs}(f,x)$ and $\mathrm{FC}(f,x)$ are dual linear programs, hence their optimal values are equal, $\mathrm{fbs}(f,x) = \mathrm{FC}(f,x)$. As an immediate corollary, $\mathrm{fbs}(f) = \mathrm{FC}(f)$.

2.3 One-Sided Measures

For Boolean functions with $H = \{0,1\}$, for each measure M from $\mathrm{bs}(f), \mathrm{fbs}(f), \mathrm{FC}(f), \mathrm{C}(f)$ and a Boolean value $b \in \{0,1\}$, define the corresponding one-sided measure as

$$M^b(f) = \max_{x\in f^{-1}(b)} M(f,x).$$

According to the earlier definitions, we then have $M(f) = \max\{M^0(f), M^1(f)\}$. These one-sided measures are useful when, for example, working with compositions of Or with some Boolean function.

2.4 Kolmogorov Complexity

A set of strings $\mathcal{S} \subset \{0,1\}^*$ is called *prefix-free* if there are no two strings in $\mathcal{S}$ such that one is a proper prefix of the other. Let M be a universal Turing machine and fix a prefix-free set $\mathcal{S}$. The prefix-free *Kolmogorov complexity* of x given y, is defined as the length of the shortest program from $\mathcal{S}$ that prints x when given y:

$$K(x|y) = \min\{|P| \mid P \in \mathcal{S}, M(P,y) = x\}.$$

For a detailed introduction on Kolmogorov complexity, we refer the reader to [15].

3 Classical Adversary Bounds

Let $f : S \to H$ be a function, where $S \subseteq G^n$. The following are all known to be lower bounds on bounded-error randomized query complexity.

3.1 Relational Adversary Bound

Let $R : S \times S \to \mathbb{R}_{\geq 0}$ be a real-valued function such that $R(x,y) = R(y,x)$ for all $x,y \in S$ and $R(x,y) = 0$ whenever $f(x) = f(y)$. Then for $x \in S$ and an index i, let[2]

$$\theta(x,i) = \frac{\sum_{y\in S} R(x,y)}{\sum_{y\in S: x_i\neq y_i} R(x,y)},$$

[2] We take the reciprocals of the expressions, compared to Aaronson's definition.

where $\theta(x,i)$ is undefined if the denominator is 0. Denote[3]

$$\mathrm{CRA}(f) = \max_{R} \min_{\substack{x,y\in S, i\in[n]:\\ R(x,y)>0, x_i\neq y_i}} \max\{\theta(x,i),\theta(y,i)\}.$$

See [1] for details.

3.2 Rank-1 Relational Adversary Bound

We introduce the following restriction of the relational adversary bound. Let R' be any $|S|\times|S|$ matrix of rank 1, such that:

- There exist $u,v : S\to\mathbb{R}_{\geq 0}$ such that $R'(x,y)=u(x)v(y)$ for all $x,y\in S$.
- $R'(x,y)=0$ whenever $f(x)=f(y)$.

Then set $R(x,y)=\max\{R'(x,y),R'(y,x)\}$.

Let $X=\{x \mid u(x)>0\}$ and $Y=\{y \mid v(y)>0\}$. Note that for every $x\in S$, either $u(x)$ or $v(x)$ must be 0, as $R(x,x)$ must be 0, therefore $X\cap Y=\varnothing$. Then denote

$$\mathrm{CRA}_1(f) = \max_{u,v} \min_{\substack{x\in X, y\in Y, i\in[n]:\\ u(x)v(y)>0, x_i\neq y_i}} \max\{\theta(x,i),\theta(y,i)\}.$$

where $\theta(x,i)$ can be simplified to

$$\theta(x,i)=\frac{\sum_{y\in Y} v(y)}{\sum_{y\in Y: x_i\neq y_i} v(y)} \qquad \text{and} \qquad \theta(y,i)=\frac{\sum_{x\in X} u(x)}{\sum_{x\in X: x_i\neq y_i} u(x)}.$$

Naturally, $\mathrm{CRA}_1(f)\leq\mathrm{CRA}(f)$.

As $R(x,y)=0$ whenever $f(x)=f(y)$, we have that for every output $h\in H$ either $f^{-1}(h)\cap X=\varnothing$ or $f^{-1}(h)\cap Y=\varnothing$. Therefore, $\mathrm{CRA}_1(f)$ effectively bounds the complexity of differentiating between two non-overlapping sets of outputs. This leads to the following equivalent definition for $\mathrm{CRA}_1(f)$:

▶ **Proposition 1.** *Let $A\cup B=H$ be a partition of the output alphabet, i.e., $A\cap B=\varnothing$. Let p and q be probability distributions over $X:=f^{-1}(A)$ and $Y:=f^{-1}(B)$, respectively. Then*

$$\mathrm{CRA}_1(f) = \max_{\substack{A,B\\ p,q}} \min_{\substack{i\in[n],\\ g_1,g_2\in G: g_1\neq g_2\\ \exists x\in X, y\in Y: p(x)q(y)>0}} \frac{1}{\min\{\Pr_{x\sim p}[x_i\neq g_1],\Pr_{y\sim q}[y_i\neq g_2]\}}.$$

For the proof of this proposition, see [6].

3.3 Weighted Adversary Bound

Let w,w' be weight schemes as follows.

- Every pair $(x,y)\in S^2$ is assigned a non-negative weight $w(x,y)=w(y,x)$ such that $w(x,y)=0$ whenever $f(x)=f(y)$.
- Every triple (x,y,i) is assigned a non-negative weight $w'(x,y,i)$ such that $w'(x,y,i)=0$ whenever $x_i=y_i$ or $f(x)=f(y)$, and $w'(x,y,i), w'(y,x,i)\geq w(x,y)$ for all x,y,i such that $x_i\neq y_i$.

[3] One can show that there exist optimal solutions for R, thus we can maximize over R instead of taking the supremum.

For all x, i, let $wt(x) = \sum_{y \in S} w(x,y)$ and $v(x,i) = \sum_{y \in S} w'(x,y,i)$. Denote

$$\mathrm{CWA}(f) = \max_{w,w'} \min_{\substack{x,y \in S, i \in [n] \\ w(x,y) \neq 0, x_i \neq y_i}} \max \left\{ \frac{wt(x)}{v(x,i)}, \frac{wt(y)}{v(y,i)} \right\}.$$

This adversary method is formulated in [14] and is an adaptation of Ambainis' quantum adversary method [5].

3.4 Kolmogorov Complexity

Let $\sigma \in \{0,1\}^*$ be any finite string.[4] Denote

$$\mathrm{CKA}(f) = \min_{\sigma} \max_{\substack{x,y \in S \\ f(x) \neq f(y)}} \frac{1}{\sum_{i: x_i \neq y_i} \min\{2^{-K(i|x,\sigma)}, 2^{-K(i|y,\sigma)}\}}.$$

See [14] for details.

3.5 Minimax over probability distributions

Let $\{p_x\}_{x \in S}$ be a set of probability distributions over $[n]$. Denote

$$\mathrm{CMM}(f) = \min_{p} \max_{\substack{x,y \in S \\ f(x) \neq f(y)}} \frac{1}{\sum_{i: x_i \neq y_i} \min\{p_x(i), p_y(i)\}}.$$

See [14] for details.

4 Equivalence of the Adversary Bounds

In this section we prove the main theorem:

▶ **Theorem 2.** *Let $f : S \to H$ be a partial Boolean function, where $S \subseteq G^n$. Then*

- $\mathrm{fbs}(f) \leq \mathrm{CRA}_1(f) \leq \mathrm{CRA}(f) = \mathrm{CWA}(f)$,
- $\mathrm{CWA}(f) = O(\mathrm{CKA}(f))$,
- $\mathrm{CKA}(f) = \Theta(\mathrm{CMM}(f))$.

Moreover, for total functions $f : G^n \to H$, we have $\mathrm{fbs}(f) = \mathrm{CMM}(f)$.

The part $\mathrm{CWA}(f) = O(\mathrm{CKA}(f))$ has been already proven in [14].

4.1 Fractional Block Sensitivity and the Weighted Adversary Method

First, we prove that fractional block sensitivity lower bounds the relational adversary bound for any partial function.

▶ **Proposition 3.** *Let $f : S \to H$ be a partial Boolean function, where $S \subseteq G^n$. Then* $\mathrm{fbs}(f) \leq \mathrm{CRA}_1(f)$.

[4] By the argument of [19], we take the minimum over the strings instead of the algorithms computing f.

Proof. Let $x \in S$ be such that $\mathrm{fbs}(f,x) = \mathrm{fbs}(f)$ and denote $h = f(x)$. Let $H' = H \setminus \{h\}$ and $S' = f^{-1}(H')$.

Let $\mathcal{B}$ be the set of sensitive blocks of x. Let $w : \mathcal{B} \to [0,1]$ be an optimal solution to the $\mathrm{fbs}(f,x)$ linear program, that is, $\sum_{B\in\mathcal{B}} w(B) = \mathrm{fbs}(f,x)$. For each $B \in \mathcal{B}$, pick a single $y_B \in S'$ such that $B = \{i \mid x_i \neq y_i\}$. Then define $R(x, y_B) := w(B)$ for all $B \in \mathcal{B}$. It is clear that R has a corresponding rank 1 matrix R', as it has only one row (corresponding to x) that is not all zeros.

Let $y \in S'$ be any input such that $R(x,y) > 0$. Then for any $i \in [n]$ such that $x_i \neq y_i$,

$$\theta(x,i) = \frac{\sum_{B\in\mathcal{B}} w(B)}{\sum_{B\in\mathcal{B}:i\in B} w(B)} = \frac{\mathrm{fbs}(f,x)}{\sum_{B\in\mathcal{B}:i\in B} w(B)} \geq \mathrm{fbs}(f),$$

as $0 < \sum_{B\in\mathcal{B}:i\in B} w(B) \leq 1$. On the other hand, note that $\theta(y,i) = \frac{w(B)}{w(B)} = 1$, where $B = \{i \mid x_i \neq y_i\}$. Therefore, for this R,

$$\min_{\substack{x,y\in S, i\in[n]:\\ R(x,y)>0, x_i\neq y_i}} \max\{\theta(x,i), \theta(y,i)\} \geq \min_{\substack{y\in S', i\in[n]:\\ R(x,y)>0, x_i\neq y_i}} \max\{\mathrm{fbs}(f), 1\} = \mathrm{fbs}(f),$$

and the claim follows. ◀

As mentioned in [14], $\mathrm{CRA}(f)$ is a weaker version of $\mathrm{CWA}(f)$. We show that in fact they are exactly equal:

▶ **Proposition 4.** *Let $f : S \to H$ be a partial Boolean function, where $S \subseteq G^n$. Then* $\mathrm{CRA}(f) = \mathrm{CWA}(f)$.

Proof.

- First we show that $\mathrm{CRA}(f) \leq \mathrm{CWA}(f)$.
  Suppose that R is the function for which the relational bound achieves maximum value. Let $w(x,y) = w(y,x) = w(x,y,i) = w(y,x,i) = R(x,y)$ for any x, y, i such that $f(x) \neq f(y)$ and $x_i \neq y_i$. This pair of weight schemes satisfies the conditions of the weighted adversary bound. The value of the latter with w, w' is equal to $\mathrm{CRA}(f)$. As the weighted adversary bound is a maximization measure, $\mathrm{CRA}(f) \leq \mathrm{CWA}(f)$.
- Now we show that $\mathrm{CRA}(f) \geq \mathrm{CWA}(f)$.
  Let w, w' be optimal weight schemes for the weighted adversary bound. Let $R(x,y) = w(x,y)$ for any $x, y \in S$ such that $f(x) \neq f(y)$. Let $S' = f^{-1}(H \setminus f(x))$. Then

  $$\theta(x,i) = \frac{\sum_{y\in S'} R(x,y)}{\sum_{y\in S':x_i\neq y_i} R(x,y)} = \frac{\sum_{y\in S'} w(x,y)}{\sum_{y\in S':x_i\neq y_i} w(x,y)} \geq \frac{\sum_{y\in S'} w(x,y)}{\sum_{y\in S':x_i\neq y_i} w'(x,y,i)} = \frac{wt(x)}{v(x,i)},$$

  as $w'(x,y,i) \geq w(x,y)$ by the properties of w, w'. Similarly, $\theta(y,i) \geq \frac{wt(y)}{v(y,i)}$. Therefore, for any $x, y \in S$ and $i \in [n]$ such that $f(x) \neq f(y)$ and $x_i \neq y_i$, we have

  $$\max\{\theta(x,i), \theta(y,i)\} \geq \max\left\{\frac{wt(x)}{v(x,i)}, \frac{wt(y)}{v(y,i)}\right\}.$$

  As the relational adversary bound is a maximization measure, $\mathrm{CRA}(f) \geq \mathrm{CWA}(f)$. ◀

The proof of this proposition also shows why $\mathrm{CRA}(f)$ and $\mathrm{CWA}(f)$ are equivalent — the weight function w' is redundant in the classical case (in contrast to the quantum setting).

4.2 Kolmogorov Complexity and Minimax over Distributions

In this section we prove the equivalence between the minimax over probability distributions and Kolmogorov complexity adversary bound. It has been shown in the proof of the main theorem of [14] that $\mathrm{CMM}(f) = \Omega(\mathrm{CKA}(f))$. Here we show the other direction using a well-known result from coding theory.

▶ **Proposition 5** (Kraft's inequality). *Let S be any prefix-free set of finite strings. Then $\sum_{x\in S} 2^{-|x|} \leq 1$.*

▶ **Proposition 6.** *Let $f : S \to H$ be a partial Boolean function, where $S \subseteq G^n$. Then $\mathrm{CKA}(f) \geq \mathrm{CMM}(f)$.*

Proof. Let σ be the binary string for which $\mathrm{CKA}(f)$ achieves the smallest value. Define the set of probability distributions $\{p_x\}_{x\in S}$ on $[n]$ as follows. Let $s_x = \sum_{i\in[n]} 2^{-K(i|x,\sigma)}$ and $p_x(i) = 2^{-K(i|x,\sigma)}/s_x$. The set of programs that print out $i \in [n]$, given x and σ, is prefix-free (by the definition of $\mathcal{S}$), as the information given to all programs is the same. Thus, by Kraft's inequality, we have $s_x \leq 1$.

Examine the value of the minimax bound with this set of probability distributions. For any $x, y \in S$ and $i \in [n]$, we have

$$\min\{p_x(i), p_y(i)\} = \min\left\{\frac{2^{-K(i|x,\sigma)}}{s_x}, \frac{2^{-K(i|y,\sigma)}}{s_y}\right\} \geq \min\{2^{-K(i|x,\sigma)}, 2^{-K(i|y,\sigma)}\}.$$

Therefore, $\mathrm{CKA}(f) = \Theta(\mathrm{CMM}(f))$. ◀

4.3 Fractional Block Sensitivity and Minimax over Distributions

Now we proceed to prove that for total functions, fractional block sensitivity is equal to the minimax over probability distributions. The latter has the following equivalent form.

▶ **Lemma 7.** *For any partial Boolean function $f : S \to H$, where $S \subseteq G^n$,*

$$\mathrm{CMM}(f) = \min_{v} \max_{x\in S} \sum_{i\in[n]} v_x(i) \quad \text{s.t. } \forall y \in S \text{ s.t. } f(x) \neq f(y) : \sum_{i:x_i\neq y_i} \min\{v_x(i), v_y(i)\} \geq 1,$$

where $\{v_x\}_{x\in S}$ is any set of weight functions $v_x : [n] \to \mathbb{R}_{\geq 0}$.

For the proof of this lemma, see [6].

In this case we prove that for total functions the minimax over probability distributions is equal to the fractional certificate complexity $\mathrm{FC}(f)$. The result follows since $\mathrm{FC}(f) = \mathrm{fbs}(f)$. The proof of this claim is almost immediate in light of the following "fractional certificate intersection" lemma by Kulkarni and Tal:

▶ **Proposition 8** ([13], Lemma 6.2). *Let $f : G^n \to H$ be a total function[5] and $\{v_x\}_{x\in G^n}$ be a feasible solution for the $\mathrm{FC}(f)$ linear program. Then for any two inputs $x, y \in G^n$ such that $f(x) \neq f(y)$, we have $\sum_{i:x_i\neq y_i} \min\{v_x(i), v_y(i)\} \geq 1$.*

Let f be a total function. Suppose that $\{v_x\}_{x\in G^n}$ is a feasible solution for the $\mathrm{CMM}(f)$ program. Then for any $x, y \in G^n$ such that $f(x) \neq f(y)$, we have $\sum_{i:x_i\neq y_i} v_x(i) \geq$

[5] Kulkarni and Tal prove the lemma for Boolean functions, but it is straightforward to check that their proof also works for functions with arbitrary input and output alphabets.

$\sum_{i:x_i \neq y_i} \min\{v_x(i), v_y(i)\} \geq 1$. Hence this is also a feasible solution for the $\mathrm{FC}(f)$ linear program. On the other hand, if $\{v_x\}_{x \in G^n}$ is a feasible solution for $\mathrm{FC}(f)$ linear program, then it is also a feasible solution for the $\mathrm{CMM}(f)$ program by Proposition 8. Therefore, $\mathrm{CMM}(f) = \mathrm{FC}(f)$.

5 Separations for Partial Functions

5.1 Fractional Block Sensitivity vs. Adversary Bounds

Here we show an example of a partial function that provides an unbounded separation between the adversary measures and fractional block sensitivity.

▶ **Theorem 9.** *There exists a partial Boolean function $f : S \to \{0,1\}$, where $S \subseteq \{0,1\}^n$, such that $\mathrm{fbs}(f) = O(1)$ and $\mathrm{CRA}_1(f), \mathrm{CRA}(f), \mathrm{CWA}(f), \mathrm{CKA}(f), \mathrm{CMM}(f) = \Omega(n)$.*

Proof. Let n be an even number and $S = \{x \in \{0,1\}^n \mid |x| = 1\}$ be the set of bit strings of Hamming weight 1. Define the "greater than half" function $\textsc{Gth}_n : S \to \{0,1\}$ to be 1 iff $x_i = 1$ for $i > n/2$.

For the first part, the certificate complexity is constant $\mathrm{C}(\textsc{Gth}_n) = 1$. To certify the value of greater than half, it is enough to certify the position of the unique i such that $x_i = 1$. The claim follows, as $\mathrm{C}(f) \geq \mathrm{fbs}(f)$ for any f.

For the second part, by Theorem 2, it suffices to show that $\mathrm{CRA}_1(\textsc{Gth}_n) = \Omega(n)$. Let $X = f^{-1}(0)$ and $Y = f^{-1}(1)$. Let $R(x,y) = 1$ for all $x \in X, y \in Y$. Suppose that $x \in X, y \in Y, i \in [n]$ are such that $x_i = 1$ (and thus $y_i = 0$). Then

$$\theta(x,i) = \frac{\sum_{y^* \in Y} R(x,y^*)}{\sum_{\substack{y^* \in Y: \\ x_i \neq y^*_i}} R(x,y^*)} = \frac{n/2}{n/2} = 1, \qquad \theta(y,i) = \frac{\sum_{x^* \in X} R(x^*,y)}{\sum_{\substack{x^* \in X: \\ x^*_i \neq y_i}} R(x^*,y)} = \frac{n/2}{1} = n/2.$$

Therefore, $\max\{\theta(x,i), \theta(y,i)\} = n/2$. Similarly, if i is such an index that $y_i = 1$ and $x_i = 0$, we also have $\max\{\theta(x,i), \theta(y,i)\} = n/2$. Also note that R has a corresponding rank 1 matrix R', hence $\mathrm{CRA}_1(f) \geq n/2 = \Omega(n)$. ◀

We note that a similar function was used to prove lower bounds on the problem of inverting a permutation [4, 1]. More specifically, we are given a permutation $\sigma(1), \ldots, \sigma(n)$, and the function is 0 if $\sigma^{-1}(1) \leq n/2$ and 1 otherwise. With a single query, one can find the value of $\sigma(i)$ for any i. By construction, a lower bound on $\textsc{Gth}_n$ also gives a lower bound on computing this function.

5.2 Relational Adversary vs. Kolmogorov Complexity Bound

Here we show that, for a variant of the ordered search problem, the Kolmogorov complexity bound gives a tight logarithmic lower bound, while the relational adversary gives only a constant value lower bound.

Let $S = \{x \in \{0,1\}^n \mid \exists i \in [0;n] : x_1 = \ldots x_i = 0 \text{ and } x_{i+1} = \ldots = x_n = 1\}$. In other words, x is any string starting with some number of 0s followed by all 1s. Define the "ordered search parity" function $\textsc{Osp}_n : S \to \{0,1\}$ to be $\textsc{Ind}(x) \bmod 2$, where $\textsc{Ind}(x)$ is the last index i such that $x_i = 0$ (in the special case $x = 1^n$, assume that $i = 0$).

▶ **Theorem 10.** *For the ordered search parity, $\mathrm{CRA}_1(\textsc{Osp}_n), \mathrm{CRA}(\textsc{Osp}_n), \mathrm{CWA}(\textsc{Osp}_n) = O(1)$ and $\mathrm{CKA}(\textsc{Osp}_n), \mathrm{CMM}(\textsc{Osp}_n) = \Omega(\log n)$.*

For the proof of this theorem, see [6].

6 Limitation of Fractional Block Sensitivity

In this section we show that there is a certain barrier that the fractional block sensitivity cannot overcome for partial functions.

6.1 Upper Bound in Terms of Block Sensitivity

▶ **Theorem 11.** *For any partial function $f : S \to H$, where $S \subseteq G^n$, $\mathrm{fbs}(f) \leq \sqrt{n \cdot \mathrm{bs}(f)}$.*

Proof. We will prove that $\mathrm{fbs}(f,x) \leq \sqrt{n \cdot \mathrm{bs}(f,x)}$ for any $x \in S$. First we introduce a parametrized version of the fractional block sensitivity. Let $x \in S$ be any input, $\mathcal{B}$ the set of sensitive blocks of x and $N \leq n$ a positive real number. Define

$$\mathrm{fbs}_N(f,x) = \max_w \sum_{B\in\mathcal{B}} w(B) \quad \text{s.t.} \quad \forall i \in [n] : \sum_{B\in\mathcal{B}: i\in B} w(B) \leq 1, \quad \sum_{B\in\mathcal{B}} |B| \cdot w(B) \leq N.$$

where $w : \mathcal{B} \to [0;1]$. If we let $N = n$, then the second condition becomes redundant and $\mathrm{fbs}_n(f,x) = \mathrm{fbs}(f,x)$.

For simplicity, let $k = \mathrm{bs}(f,x)$. We will prove by induction on k that $\mathrm{fbs}_N(f,x) \leq \sqrt{Nk}$. If $k = 0$, the claim obviously holds, so assume $k > 0$. Let ℓ be the length of the shortest block in $\mathcal{B}$. Then

$$\sum_{B\in\mathcal{B}} \ell \cdot w(B) \leq \sum_{B\in\mathcal{B}} |B| \cdot w(B) \leq N$$

and $\mathrm{fbs}_N(f,x) = \sum_{B\in\mathcal{B}} w(B) \leq N/\ell$.

On the other hand, let D be any shortest sensitive block. Let f' be the restriction of f where the variables with indices in D are fixed to the values of x_i for all $i \in D$. Note that $\mathrm{bs}(f',x) \leq k-1$, as we have removed all sensitive blocks that overlap with D. Let $\mathcal{B}'$ be the set of sensitive blocks of x on f' and let $\mathcal{T} = \{B \in \mathcal{B} \mid B \cap D \neq \varnothing\}$, the set of sensitive blocks that overlap with D (including D itself). Then no $T \in \mathcal{T}$ is a member of $\mathcal{B}'$, therefore

$$\sum_{B'\in\mathcal{B}'} |B'| \cdot w(B') \leq N - \sum_{T\in\mathcal{T}} |T| \cdot w(T) \leq N - \ell \cdot \sum_{T\in\mathcal{T}} w(T).$$

Denote $t = \sum_{T\in\mathcal{T}} w(T)$. We have that $t \leq |D| = \ell$, as any $T \in \mathcal{T}$ overlaps with D. By combining the two inequalities we get

$$\begin{aligned}\mathrm{fbs}_N(f,x) &\leq \max_{\ell\in[0;n]} \min\left\{\frac{N}{\ell}, \max_{t\in[0;\ell]} \{t + \mathrm{fbs}_{N-\ell t}(f',x)\}\right\} \\ &\leq \max_{\ell\in[0;n]} \min\left\{\frac{N}{\ell}, \max_{t\in[0;\ell]} \left\{t + \sqrt{(N-\ell t)(k-1)}\right\}\right\}.\end{aligned}$$

If $N/\ell \leq \sqrt{Nk}$, we are done. Thus further assume that $\ell < \sqrt{N/k}$.

Denote $g(t) = t + \sqrt{(N-\ell t)(k-1)}$. We need to find the maximum of this function on the interval $[0;\ell]$ for a given ℓ. Its derivative, $g'(t) = 1 - \frac{\ell}{2}\sqrt{\frac{k-1}{N-\ell t}}$, is a monotone function in t. Thus, it has exactly one root, $t_0 = N/\ell - (k-1)\cdot\ell/4$. Therefore, $g(t)$ attains its maximum value on $[0;\ell]$ at one of the points $\{0, t_0, \ell\}$.

- If $t = 0$, then $g(0) = \sqrt{N(k-1)} \leq \sqrt{Nk}$.
- If $t = t_0$, then, as $t \leq \ell < \sqrt{N/k}$,

$$\begin{aligned}\sqrt{Nk} - \frac{k-1}{4}\cdot\sqrt{\frac{N}{k}} &< \frac{N}{\ell} - (k-1)\frac{\ell}{4} < \sqrt{\frac{N}{k}} \\ \sqrt{k} - \frac{k-1}{4\sqrt{k}} &< \sqrt{\frac{1}{k}}.\end{aligned}$$

Thus $3k < 0$, which has no solutions in natural numbers for k, so this case is not possible.

- If $t = \ell$, then $g(t) = \ell + \sqrt{(N-\ell^2)(k-1)}$.

Now it remains to find the maximum value of $h(k) = \ell + \sqrt{(N-\ell^2)(k-1)}$ on the interval $[0; \sqrt{N/k}]$. The derivative is equal to $h'(\ell) = 1 - \ell \cdot \sqrt{\frac{k-1}{N-\ell^2}}$. The only non-negative root of $h'(\ell)$ is equal to $\ell_0 = \sqrt{N/k}$. Then $h(\ell)$ is monotone on the interval $[0; \sqrt{N/k}]$. Thus $h(\ell)$ attains its maximal value at one of the points $\{0, \sqrt{N/k}\}$.

- If $\ell = 0$, then $h(\ell) = \sqrt{N(k-1)} < \sqrt{Nk}$.
- If $\ell = \ell_0 = \sqrt{N/k}$, then

$$h(\ell) = \sqrt{\frac{N}{k}} + \sqrt{\left(N - \frac{N}{k}\right)(k-1)} = \sqrt{N}\left(\sqrt{\frac{1}{k}} + (k-1)\sqrt{\frac{1}{k}}\right) = \sqrt{Nk}.$$

Thus, $h(\ell) \le \sqrt{Nk}$ and that concludes the induction.

Therefore, $\mathrm{fbs}(f,x) = \mathrm{fbs}_n(f,x) \le \sqrt{n \cdot \mathrm{bs}(f,x)}$, hence also $\mathrm{fbs}(f) \le \sqrt{n \cdot \mathrm{bs}(f)}$. ◀

6.2 A Matching Construction

▶ **Theorem 12.** *For any $k \in \mathbb{N}$, there exists a partial Boolean function $f : S \to \{0,1\}$, where $S \subseteq \{0,1\}^n$, such that $\mathrm{bs}(f) = k$ and $\mathrm{fbs}(f) = \Omega(\sqrt{n \cdot \mathrm{bs}(f)})$.*

Proof. Take any finite projective plane of order t, then it has $\ell = t^2 + t + 1$ many points. Let $n = k\ell$ and enumerate the points with integers from 1 to ℓ. Let $X = \{0^\ell\}$ and $Y = \{y \mid$ there exists a line L such that $y_i = 1$ iff $i \in L\}$. Define the (partial) finite projective plane function $\textsc{Fpp}_t : X \cup Y \to \{0,1\}$ as $\textsc{Fpp}_t(y) = 1 \iff y \in Y$.

We can calculate the 1-sided block sensitivity measures for this function:

- $\mathrm{fbs}^0(\textsc{Fpp}_t) \ge (t^2+t+1) \cdot \frac{1}{t+1} = \Omega(t)$, as each line gives a sensitive block for 0^n; since each point belongs to $t+1$ lines, we can assign weight $1/(t+1)$ for each sensitive block and that is a feasible solution for the fractional block sensitivity linear program.
- $\mathrm{bs}^0(\textsc{Fpp}_t) = 1$, as any two lines intersect, so any two sensitive blocks of 0^n overlap.
- $\mathrm{bs}^1(\textsc{Fpp}_t) = 1$, as there is only one negative input.

Next, define $f : S^{\times k} \to \{0,1\}$ as the composition of $\textsc{Or}$ with the finite projective plane function, $f = \textsc{Or}_k(\textsc{Fpp}_t(x^{(1)}), \ldots, \textsc{Fpp}_t(x^{(k)}))$. By the properties of composition with $\textsc{Or}$ (see Proposition 31 in [10] for details), we have

- $\mathrm{fbs}(f) = \max\{\mathrm{fbs}^0(f), \mathrm{fbs}^1(f)\} \ge \mathrm{fbs}^0(f) = \mathrm{fbs}^0(\textsc{Fpp}_t) \cdot k = \Theta(t) \cdot k = \Theta(t \cdot n/t^2) = \Theta(n/t)$,
- $\mathrm{bs}(f) = \max\{\mathrm{bs}^0(f), \mathrm{bs}^1(f)\} = \mathrm{bs}^0(\textsc{Fpp}_t) \cdot k = k = \Theta(n/t^2)$.

As $\sqrt{n \cdot n/t^2} = n/t$, we have $\mathrm{fbs}(f) = \Omega(\sqrt{n \cdot \mathrm{bs}(f)})$ and hence the result. ◀

Note that our example is also tight in regard to the multiplicative constant, since t can be unboundedly large (and the constant arbitrarily close to 1).

7 Open Ends

Rank 1 Weighted Adversary

Although we have shown that $\mathrm{CRA}(f)$ and $\mathrm{CKA}(f)$ are not equivalent for partial functions, there is still a possibility that $\mathrm{CRA}_1(f) = \Theta(\mathrm{CRA}(f))$ might be true. If they are indeed equivalent, then the weighted adversary would have a simpler formulation to use.

Limitation of the Adversary Bounds

In the quantum setting, the certificate barrier shows a limitation on the quantum adversary bounds. In the classical setting, by our results, fractional block sensitivity characterizes the classical adversary bounds for total functions and thus is of course an upper bound. Is there a general limitation on the classical adversary methods for partial functions?

Block Sensitivity vs. Fractional Block Sensitivity

We have exhibited an example with the largest separation between the two measures for partial functions, $\mathrm{bs}(f) = O(\sqrt{n \cdot \mathrm{bs}(f)})$. For total functions, one can show that $\mathrm{fbs}(f) \leq \mathrm{bs}(f)^2$, but the best known separation achieves $\mathrm{fbs}(f) = \Omega(\mathrm{bs}(f)^{3/2})$ [10]. Can our results be somehow extended for total functions to close the gap?

References

1 Scott Aaronson. Lower bounds for local search by quantum arguments. *SIAM Journal on Computing*, 35(4):804–824, 2006.

2 Scott Aaronson. Quantum certificate complexity. *Journal of Computer and System Sciences*, 74(3):313–322, 2008.

3 Scott Aaronson, Shalev Ben-David, and Robin Kothari. Separations in query complexity using cheat sheets. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC'16, pages 863–876, New York, NY, USA, 2016. ACM.

4 Andris Ambainis. Quantum lower bounds by quantum arguments. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC'00, pages 636–643, New York, NY, USA, 2000. ACM.

5 Andris Ambainis. Polynomial degree vs. quantum query complexity. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, FOCS'03, pages 230–239, Washington, DC, USA, 2003. IEEE Computer Society.

6 Andris Ambainis, Martins Kokainis, Krisjanis Prusis, and Jevgenijs Vihrovs. All classical adversary methods are equivalent for total functions. *CoRR*, abs/1709.08985, 2017. `arXiv:1709.08985`.

7 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001.

8 Shalev Ben-David and Robin Kothari. Randomized query complexity of sabotaged and composed functions. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 60:1–60:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

9 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

10 Justin Gilmer, Michael Saks, and Srikanth Srinivasan. Composition limits and separating examples for some Boolean function complexity measures. *Combinatorica*, 36(3):265–311, 2016.

11 Peter Hoyer, Troy Lee, and Robert Spalek. Negative weights make adversaries stronger. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC'07, pages 526–535, New York, NY, USA, 2007. ACM.

12 Rahul Jain and Hartmut Klauck. The partition bound for classical communication complexity and query complexity. In *Proceedings of the 2010 IEEE 25th Annual Conference on Computational Complexity*, CCC'10, pages 247–258, Washington, DC, USA, 2010. IEEE Computer Society.

13 Raghav Kulkarni and Avishay Tal. On fractional block sensitivity. *Chicago Journal Of Theoretical Computer Science*, 8:1–16, 2016.

14 Sophie Laplante and Frédéric Magniez. Lower bounds for randomized and quantum query complexity using Kolmogorov arguments. In *Proceedings of the 19th IEEE Annual Conference on Computational Complexity*, CCC'04, pages 294–304, Washington, DC, USA, 2004. IEEE Computer Society.

15 Ming Li and Paul M.B. Vitnyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Publishing Company, Incorporated, 3 edition, 2008.

16 Noam Nisan. CREW PRAMs and decision trees. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, STOC'89, pages 327–335, New York, NY, USA, 1989. ACM.

17 Noam Nisan and Mario Szegedy. On the degree of Boolean functions as real polynomials. *Computational Complexity*, 4(4):301–313, 1994.

18 Ben W. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every Boolean function. In *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS'09, pages 544–551, Washington, DC, USA, 2009. IEEE Computer Society.

19 Robert Špalek and Mario Szegedy. All quantum adversary methods are equivalent. *Theory of Computing*, 2(1):1–18, 2006.

20 Avishay Tal. Properties and applications of Boolean function composition. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS'13, pages 441–454, New York, NY, USA, 2013. ACM.

21 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science*, pages 222–227, 1977.

Computing Hitting Set Kernels by AC^0-Circuits

Max Bannach
Institute of Theoretical Computer Science, Universität zu Lübeck, Lübeck, Germany
bannach@tcs.uni-luebeck.de

Till Tantau
Institute of Theoretical Computer Science, Universität zu Lübeck, Lübeck, Germany
tantau@tcs.uni-luebeck.de

Abstract

Given a hypergraph $H = (V, E)$, what is the smallest subset $X \subseteq V$ such that $e \cap X \neq \emptyset$ holds for all $e \in E$? This problem, known as the *hitting set problem,* is a basic problem in parameterized complexity theory. There are well-known kernelization algorithms for it, which get a hypergraph H and a number k as input and output a hypergraph H' such that (1) H has a hitting set of size k if, and only if, H' has such a hitting set and (2) the size of H' depends only on k and on the maximum cardinality d of edges in H. The algorithms run in polynomial time, but are highly sequential. Recently, it has been shown that one of them can be parallelized to a certain degree: one can compute hitting set kernels in parallel time $O(d)$ – but it was conjectured that this is the best parallel algorithm possible. We refute this conjecture and show how hitting set kernels can be computed in *constant* parallel time. For our proof, we introduce a new, generalized notion of hypergraph sunflowers and show how iterated applications of the color coding technique can sometimes be collapsed into a single application.

2012 ACM Subject Classification Mathematics of computing → Hypergraphs, Theory of computation → Fixed parameter tractability, Theory of computation → Circuit complexity

Keywords and phrases Parallel Computation, Fixed-parameter Tractability, Kernelization

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.9

Related Version Full proofs can be found in the full version of the paper [5], `http://arxiv.org/abs/1801.00716`.

1 Introduction

The hitting set problem is the following combinatorial problem: Given a hypergraph $H = (V, E)$ as input, consisting of a set V of vertices and a set E of *hyperedges* with $e \subseteq V$ for all $e \in E$, find a set $X \subseteq V$ of minimum size that "hits" all hyperedges $e \in E$, that is, $e \cap X \neq \emptyset$. Many problems reduce to the hitting set problem, including the vertex cover problem (it is exactly the special case where all edges have size $|e| = 2$) and the dominating set problem (a dominating set of a graph is exactly a hitting set of the hypergraph whose hyperedges are the closed neighborhoods of the graph's vertices). The computational complexity of the hitting set problem is thus of interest both in classical complexity theory and in parameterized complexity theory.

The first result on the parameterized complexity of the hitting set problem was an efficient *kernelization algorithm* for this problem restricted to edges of cardinality three [16]. This was later improved to a kernelization for the d-uniform version (all hyperedges have size exactly d) [15], which is based on the so-called Sunflower Lemma [13]. We will later have a closer look at this algorithm; at this point let us just summarize its main idea by "repeatedly find sunflowers and replace them by their cores until there are no more sunflowers." The

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 9; pp. 9:1–9:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Sunflower Lemma tells us that this algorithm will stop only when the input graph has been reduced to a kernel. The just-sketched kernelization algorithm is highly sequential, but Chen et al. [11] have recently shown that it can be parallelized: Instead of reducing sunflowers one-at-a-time, one can replace all sunflowers in a hypergraph by their cores simultaneously in constant parallel time. This process only needs to be repeated $d(H) = \max_{e \in E} |e|$ times, leading to a parallel algorithm running in time $O(d(H))$. However, there were good reasons to believe that this algorithm is essentially the best possible (we will later discuss them) and Chen et al. conjectured that the hitting set problem does not admit a kernelization algorithm running in constant parallel time (that is, in time completely independent of the input graph).

Our Contributions. In the present paper we refute the conjecture of Chen et al. and show that there is a constant parallel time kernelization algorithm for the hitting set problem:

▶ **Problem 1.1.** $p_{k,d}$-HITTING-SET
Instance: *A hypergraph $H = (V, E)$ and a number $k \in \mathbb{N}$.*
Parameter: $k + d(H)$
Question: *Does H have a hitting set X with $|X| \leq k$?*

▶ **Theorem 1.2** (Main Theorem)**.** *There is a* DLOGTIME*-uniform* AC^0*-circuit family that maps every hypergraph $H = (V, E)$ and number k to a new hypergraph $H' = (V, E')$ that has the same size-k hitting sets as H, has $d(H') \leq d(H)$, and has $|E'| \leq f(k, d(H))$ for some fixed computable function f.*

Let us stress at this point that the AC^0-family from the theorem really has a size that is polynomial in the input length (no exponential or even worse dependency on the parameters) and has a depth that is completely independent of the input. The hypergraph H' has the same vertex set V as H – a feature shared by all hypergraphs considered in this paper that simplifies the presentation. However, since V is still "large," the circuit is not quite a kernelization algorithm. Fortunately, this is easy to fix by replacing the vertex set of H' by $V' = \bigcup_{e \in E'} e$, yielding the following corollary:

▶ **Corollary 1.3** (Constant-Time Kernelization)**.** *There is a* DLOGTIME*-uniform* AC^0*-circuit family that computes a kernel for every instance for $p_{k,d}$-*HITTING-SET*.*

The theorem and corollary imply that all problems that can be reduced to $p_{k,d}$-HITTING-SET via a parameter-preserving AC^0-reduction admit a kernelization computable by an AC^0-circuit family. This includes p_k-VERTEX-COVER, which is just $p_{k,d}$-HITTING-SET with d fixed at 2; p_k-TRIANGLE-REMOVAL, where the objective is to remove at most k vertices from an undirected graph so that no triangles remain; and also $p_{k,\deg}$-DOMINATING-SET, where we must find a dominating set of size at most k in an undirected graph and we parameterized by k and the maximum degree of the vertices.

Our proof of the main theorem requires the development of two new ideas, which we believe may also be useful in other situations. The above-mentioned parallel kernelization algorithm for the hitting set problem with runtime $O(d(H))$ essentially does the following: "Repeat $d(H)$ times: replace all sunflowers of size $k + 1$ by their cores" and the difficult task in each of the $d(H)$ iterations is to find the sunflowers. It turns out that this can be done in constant parallel time using the *color coding* technique [2] and it has been shown in [3] and again in [11] that this technique can be implemented in constant time. Our first idea for turning the circuits depth from $O(d)$ into $O(1)$ is to *collapse the color codings from the d rounds into a single application of the color coding technique:* Instead of applying color

coding in each round to filter and describe "objects," we would like to apply one global application of color coding that already contains the internal colorings and does away with the intermediate objects.

Unfortunately, there does not appear to be a simple (or any) way of actually collapsing the colorings used when we "replace all sunflowers by their cores": The coloring coding technique is good at imposing requirements of the form "these objects must be disjoint," but cannot impose requirements of the form "these objects must be the same." For this reason, as our second new idea, we develop a generalization of the notion of a sunflower (which we dub "pseudo-sunflowers") that is tailored to the collapsing of color coding.

Related Work. The sequential kernelization algorithm for the hitting set problem based on the Sunflower Lemma has been known for a longer time [15], but there have been recent improvements that bring down the runtime to linear time [17]. A parallel version has recently been studied by Chen et al. [11] and they show how kernels for $p_{k,d}$-HITTING-SET can be computed by circuits of depth $O(d(H))$. Chen et al. also conjecture that the circuit depth of $O(d(H))$ is unavoidable (which we refute).

The results of this paper fit into the larger, fledgling field of parallel parameterized complexity theory, which has already been studied both from a practical [1] and a theoretical point of view [8]. First results go back to research on *parameterized logarithmic space* [7, 10, 14], since it is known from classical complexity theory that problems that are solvable with such a resource bound can also be parallelized. A more structured analysis of parameterized space and circuit classes was later made by Elberfeld et al. [12], which addresses parallelization more directly. Current research on parameterized parallelization – including this paper – focuses on constant-time computations, that is, on a parameterized analogue of AC^0 [9, 11, 3, 4]. We remark that many previous results (including several of the authors) boil down to showing that instead of using a known reduction rule many times sequentially, one can simply apply it in parallel "everywhere," but "only once." In contrast, the kernelization algorithm developed in the present paper had no previous counterpart in the sequential setting.

Organization of This Paper. After a short section on preliminaries, in Section 3 we review known kernelization algorithms for the hitting set problem – both the sequential ones and the parallel one. In Section 4 we discuss the obstacles that must be surmounted to turn the known parallel algorithm into one that needs only constant time. Towards this aim, we introduce the notions of pseudo-cores and pseudo-sunflowers as replacements for the cores and sunflowers used in the known algorithms. In Section 5 we then argue that these pseudo-sunflowers can be computed in constant time by "collapsing" multiple rounds of color coding into a single round. Full proofs can be found in the full version of the paper [5].

2 Preliminaries

A *hypergraph* is a pair $H = (V, E)$ such that for all *hyperedges* $e \in E$ we have $e \subseteq V$. We write $V(H) = V$ and $E(H) = E$ for the vertex and hyperedge sets of H. Let $d(H) = \max_{e \in E} |e|$. Throughout this paper, all hypergraphs will always have the same vertex set V, which is the input vertex set. For this reason, in slight abuse of notation, for two hypergraphs $H_1 = (V, E_1)$ and $H_2 = (V, E_2)$ we also write $H_1 \subseteq H_2$ for $E(H_1) \subseteq E(H_2)$ and $H_1 \cup H_2$ for $(V, E(H_1) \cup E(H_2))$.

Concerning circuit classes and parallel computations, we will only need the notion of AC-circuit families, which are sequences $C = (C_0, C_1, C_2, \dots)$ of Boolean circuits where each

C_i is a directed acyclic graph whose vertices are gates such that there are i input gates, the inner gates are $\wedge$-gates or $\vee$-gates with unbounded fan-in, or $\neg$-gates; and the number of output gates is either 1 (for decision problems) or depends on the number of input gates (for circuits computing a function). The *size function* S maps circuits to their size (number of gates) and the *depth function* D maps them to their depth (longest path from input gates to output gates). When $D(C_n) \in O(1)$ and $S(C_n) \in n^{O(1)}$ hold, we call C an AC^0-circuit family. Concerning circuit uniformity, all circuit families in this paper will be DLOGTIME uniform, which is the strongest notion of uniformity commonly considered [6] and defined as follows: there is a DTM that on input of $\text{bin}(i)\#\,\text{bin}(n)$, where $\text{bin}(x)$ is the binary encoding of x, outputs the ith bit of a suitable encoding of C_n in at most $O(\log n)$ steps.

Even though this paper is about a parallel kernelization algorithm, we will need only little from the machinery of parallel parameterized complexity theory. We do need the following notions: A *parameterized problem* is a pair (Q, κ) where $Q \subseteq \Sigma^*$ is a language and κ is a function $\kappa\colon \Sigma^* \to \mathbb{N}$ that is computable by a DLOGTIME-uniform AC^0-circuit family. When we write down a parameterized problem such as $p_{k,d}$-HITTING-SET, the indices of "p" (for "parameterized") indicate which parameter function κ we mean. A *kernelization* for a parameterized problem (Q, κ) is a function K that maps every instance $x \in \Sigma^*$ to a new instance $K(x) \in \Sigma^*$ such that for all $x \in \Sigma^*$ we have (1) $x \in Q \iff K(x) \in Q$ and (2) $|K(x)| \leq f(\kappa(x))$ for some fixed computable function f.

A parameterized problem (Q, κ) lies in FPT if $x \in Q$ can be decided by a sequential algorithm running in time $f(\kappa(x)) \cdot |x|^{O(1)}$ for a computable function f. The AC^0-analogue of FPT is the class para-AC^0. It contains all problems (Q, κ) for which there is a circuit family $(C_{n,k})_{n,k\in\mathbb{N}}$ such that for all inputs x we have $C_{|x|,\kappa(x)}(x) = 1$ if, and only if, $x \in Q$, and $D(C_{n,k}) \in O(1)$ and $S(C_{n,k}) \in f(k) \cdot n^{O(1)}$. It is well-known that $(Q, \kappa) \in$ FPT holds if, and only if, Q is decidable and there is a kernelization for (Q, κ) that is computable in polynomial time. The same proof as for the polynomial-time case also shows that we have $(Q, \kappa) \in$ para-AC^0 if, and only if, Q is decidable and (Q, κ) has a kernelization that can be computed by an AC^0-circuit family. (We stress once more that this means that the kernelization is a normal AC^0-circuit family, having size $S(C_n) \in n^{O(1)}$.)

We will use the *color coding technique* a lot. First introduced in [2], it has recently been shown to work in the context of constant time computations [3, 11]. The key observation underlying this technique is the following: Suppose we are given a set of n elements and suppose you have k special elements $x_1, \ldots, x_k$ together with some specific colors $c_1, \ldots, c_k$ for them "in mind". Then we can compute a set Λ of "candidate colorings" of all elements of the set such that at least one $\lambda \in \Lambda$ colors each "in mind" vertex x_i with the "desired" color c_i, that is $\lambda(x_i) = c_i$. Formally, the following holds (the original version of this lemma due to Alon et al [2] is equivalent to the statement below – only without any depth guarantees):

▶ **Fact 2.1** (Color Coding Lemma, [3]). *There is a* DLOGTIME*-uniform family* $(C_{n,k,c})_{n,k,c\in\mathbb{N}}$ *of* AC*-circuits without inputs such that each* $C_{n,k,c}$

1. *outputs a set* Λ *of functions* $\lambda\colon \{1, \ldots, n\} \to \{1, \ldots, c\}$ *(coded as a sequence of function tables) with the property that for any* k *mutually distinct* $x_1, \ldots, x_k \in \{1, \ldots, n\}$ *and any* $c_1, \ldots, c_k \in \{1, \ldots, c\}$ *there is a function* $\lambda \in \Lambda$ *with* $\lambda(x_i) = c_i$ *for all* $i \in \{1, \ldots, k\}$,
2. *has constant depth (independent of* n*,* k*, or* c*), and*
3. *has size at most* $O(\log c \cdot c^{k^2} \cdot k^4 \cdot n \log^2 n)$.

3 Known Kernelization Algorithms for the Hitting Set Problem

Known Sequential Kernelization Algorithms. Known algorithms for computing kernels for $p_{k,d}$-HITTING-SET are based on the so-called *Sunflower Lemma.* The perhaps simplest application of this lemma is to repeatedly collapses sufficiently large sunflowers to their cores until there are no longer any large sunflowers in the graph and, then, the Sunflower Lemma tells us that the graph "cannot be very large." In detail, the definitions and algorithm are as follows:

▶ **Definition 3.1** (Sunflower). A *sunflower S with core C* is a set of proper supersets of C such that for any two distinct $p, q \in S$ we have $p \cap q = C$. The elements of a sunflower are called *petals.* A *sunflower in a hypergraph* is a sunflower whose petals are hyperedges of the hypergraph.

▶ **Fact 3.2** (Sunflower Lemma [13]). *Every hypergraph H with more than $k^{d(H)} \cdot d(H)!$ hyperedges contains a sunflower of size $k+1$.*

The importance of the Sunflower Lemma for the hitting set problem lies in the following observation: Suppose a hypergraph H contains a sunflower S of size at least $k+1$. Then H has a size-k hitting set if, and only if, the hypergraph obtained from H by removing all petals of the sunflower and adding its core has such a hitting set (we cannot hit the $k+1$ petals in the sunflower using only k vertices without using at least one vertex of the core; thus, we hit all petals if, and only if, we hit the core). In other words, replacing a sunflower of size $k+1$ by its core is a reduction rule for the hitting set problem; and if we can no longer apply this rule, the Sunflower Lemma tells us that the hypergraph's size is bounded by a function that depends only on k and $d(H)$ – in other words, it is a kernel.

The just-described kernelization algorithm is simple, but "very sequential." It is, however, not too difficult to turn it into a more parallel algorithm – at least, as long as $d(H)$ is fixed. This was first noted by Chen et al. [11] and we explain the ideas behind their proof below, rephrased for the purposes of the present paper.

A better sequential kernelization algorithm has recently [17] been proposed (it runs in time $O(2^{d(H)}|E|)$, which is linear from a parameterized point of view) – but the algorithm is arguably "even more sequential" and does not lend itself to easy parallelization.

Known Parallel Kernelization Algorithm. The first step towards a parallel kernelization is the observation that we can compute many cores in parallel. Given a hypergraph $H = (V, E)$ and a number k, let a *k-core in H* be a core C of a sunflower in H with more than k petals. Let $k\text{-cores}(H) = \big(V, \{C \mid C \text{ is a } k\text{-core in } H\}\big)$. While in the sequential algorithm we always replace one sunflower by its core, we now replace *all* sunflowers by their cores. This leaves behind some hyperedges, but the Sunflower Lemma will show that their number is "small." Unfortunately, the set of cores itself may still be large and we need to apply the replace-all-sunflowers-by-cores operation repeatedly. This process *does* stop after at most $d(H)$ rounds since the *size* of the cores decreases by 1 in each round and, hence, after $d(H)$ rounds it has shrunk to 0.

Let us now formalize these ideas a bit: Let $H_0 = H$ and let $H_{i+1} = k\text{-cores}(H_i)$. Then H_0 is the original hypergraph; H_1 is the set of its k-cores; H_2 is the set of H_1's k-cores and thus the set of "cores of cores" of H; next H_3 is the set of "cores of cores of cores" of H; and so on, see Figure 1 for an example. In a sense, each H_i is nested into the previous hypergraph, leading to a whole sequence resembling a matryoshka doll. Below, we define a

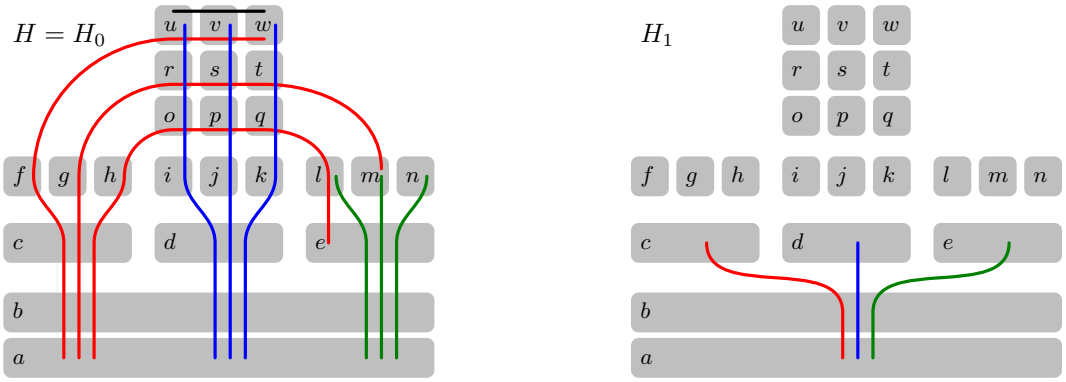


Figure 1 Visualization of a hypergraph H_0 and of its 2-cores $H_1 = 2\text{-cores}(H_0)$. Vertices are drawn as rectangles, while the ten hyperedges of H_0 are drawn as lines: they contain all vertices that they touch. For instance, the leftmost line starting in the vertex a in H_0 visualizes the hyperedge $\{a, b, c, f, u, v, w\}$ and the rightmost line visualizes the hyperedge $\{a, b, e, n\}$. The hypergraph H_0 contains three sunflowers of size 3, visualized by the red, blue, and green lines, respectively. Their cores are the hyperedges shown in H_1. These cores, in turn, form a sunflower in H_1 with core $\{a, b\}$, but note that $\{a, b\}$ is *not* a 2-core of H_0. It is the only hyperedge of H_2.

matryoshka sequence as a sequence that has this "nested in some sense" property and then show in Lemma 3.4 that $(H_0, H_1, \dots)$ is, indeed, such a matryoshka sequence:

▶ **Definition 3.3** (Matryoshka Sequence). A *matryoshka sequence for a hypergraph* $H = (V, E)$ *and a number* k is a sequence $(M_0, M_1, \dots, M_{d(H)})$ of hypergraphs, all of which have the same vertex set V, with the following properties for all $i \in \{0, \dots, d(H)\}$:

1. $M_0 = H$,
2. $d(M_i) \leq d(H) - i$,
3. $k\text{-cores}(M_i) \subseteq M_{i+1}$, and
4. every size-k hitting set of H is also a hitting set of M_i.

▶ **Lemma 3.4** (Cores of Cores Form a Matryoshka Sequence). *For every hypergraph* H *and number* k*, the sequence* $(H_0, \dots, H_{d(H)})$ *is a matryoshka sequence for* H *and* k*.*

Sketch of Proof. The first three items follow directly from the definition. The fourth item is proven by induction over i, where the inductive step hinges on the observation that the only way to hit a sunflower of size $k+1$ with a size k set X is to hit its core. ◀

Recall that the idea behind the parallel computation of a kernel for the hitting set problem is to repeatedly remove all sunflowers from H, each time perhaps leaving a manageable number of hyperedges – and after d rounds, no hyperedges will remain. We use the following notation for the "removal" operation: For two hypergraphs $H = (V, E)$ and $H' = (V, E')$ let $H \ominus H' = \big(V, \{\, e \in E \mid \forall e' \in E' \colon e' \not\subseteq e \,\}\big)$, that is, we remove all hyperedges from H that contain a hyperedge of H'. Thus, $H \ominus H_1$ is the set of all hyperedges in H that are not involved in any sunflower of size at least $k+1$ since we remove all edges that contain a core.

The following theorem shows that the repeated removing operation only leaves behind a "small" number of hyperedges. We formulate the theorem for arbitrary matryoshka sequences (we will need this later on), but it is best to think of the M_i as the sets H_i.

▶ **Theorem 3.5** (Kernel Theorem). *Let* $(M_0, \dots, M_{d(H)})$ *be a matryoshka sequence for* H *and* k*. Let* $K = (M_0 \ominus M_1) \cup (M_1 \ominus M_2) \cup (M_2 \ominus M_3) \cup \dots \cup (M_{d(H)-1} \ominus M_{d(H)}) \cup M_{d(H)}$.

1. *Then K has at most $\sum_{i=0}^{d(H)} k^i i!$ hyperedges and*
2. *H and K have the same size-k hitting sets.*

Sketch of Proof. For the first item we observe that $M_i \ominus M_{i+1}$ does not contain a sunflower and apply the Sunflower Lemma. The second item is proven by induction over i, where the base case is given by the first property of a matryoshka sequence, and where the inductive step can be derived from the fourth property of a matryoshka sequence. ◀

Instantiating the theorem with $(H_0, \ldots, H_{d(H)})$ tells us that, if we can compute the elements of $K = (H_0 \ominus H_1) \cup \cdots \cup (H_{d(H)-1} \ominus H_{d(H)}) \cup H_{d(H)}$ in parallel, we can compute a kernel for the hitting set problem in parallel. Clearly, "computing K" essentially boils down to "computing the H_i" in parallel. Thus, the real question, which we address next, is how quickly and easily we can compute the hypergraphs H_i.

At this point, we briefly need to address some technical issues concerning the coding of hypergraphs. For our purposes, it is largely a matter of taste how the input hypergraph H_0 is encoded, but the encoding of the later graphs H_i becomes important in the context of parallel constant-time computations. We consider $H = (V, E)$ fixed and encoded using, for instance, an incidence matrix (having $|V|$ columns and $|E|$ rows). We encode a *refinement of H*, that is, a hypergraph $H' = (V, E')$ with the property that each $e' \in E'$ is a subset of some $e \in E$, using a matrix of $2^{d(H)}$ columns and $|E|$ rows. There is a column for each of the at most $2^{d(H)}$ possible subsets of an edge $e \in E$ and the entry at the column for a given row is 1 if this subset is an element of E'; otherwise it is 0. Let us call this the *refinement matrix encoding* of hypergraph H' (with respect to the fixed input hypergraph H).

▶ **Lemma 3.6** (Computing Cores in Constant Depth). *For each d and i there is a* DLOGTIME*-uniform family of* AC*-circuits that*

1. *on input of the incidence matrix of a hypergraph H with $d(H) \leq d$, a number k, and the refinement matrix encoding of the hypergraph H_i,*
2. *outputs the refinement matrix encoding of H_{i+1},*
3. *has constant depth, and*
4. *has size $f(k,d) \cdot |V|^{O(1)} |E|^{O(1)}$ where f is some computable function.*

Sketch of Proof: We can test all $|E| \cdot 2^{d(H)}$ possible cores C in parallel and, for each of them, we can search a corresponding sunflower via color coding: for each petal p_i the vertices in $p_i - C$ should receive color i. ◀

The lemma tells us that once we have computed some H_i, we can compute the next H_{i+1} using only constant additional depth and using $f(k,d) \cdot |V|^{O(1)} |E|^{O(1)}$ additional size. Since $H_i \ominus H_{i+1}$ can easily be computed from H_i and H_{i+1} in constant depth, we get:

▶ **Theorem 3.7** (Depth-$O(d)$ Kernelization Algorithm, [11]). *For each d there is a* DLOGTIME*-uniform family of* AC*-circuits that*

1. *on input of a hypergraph H with $d(H) \leq d$ and a number k*
2. *outputs a hypergraph K having the same size-k hitting sets as H and having at most $\sum_{i=0}^{d(H)} k^i i!$ hyperedges,*
3. *has depth $O(d)$,*
4. *and has size $f(k,d) \cdot |V|^{O(1)} |E|^{O(1)}$ where f is some computable function.*

4 Pseudo-Cores and Pseudo-Sunflowers

The parallel kernelization algorithm described in the previous section has a depth that is linear in the parameter d, the maximum size of any hyperedge in the input hypergraph. The reason for this linear dependency was that, while we managed to reduce not just one but all sunflowers in the hypergraph to their cores in parallel, we had to repeat this "reduce to core" procedure d times – and each round adds a constant number of layers to the circuit.

It is not obvious how this build-up of layers can be avoided. In the following, we first explain why there are good reasons to believe that the computation of the hypergraphs H_i necessitates deeper and deeper circuits. Following this discussion, we explain our proposal for side-stepping these difficulties: we replace the hypergraphs H_i by new hypergraphs H'_i that are easier to compute but still form a matryoshka sequence and – hence – can serve as a replacement for the H_i in the Kernel Theorem, Theorem 3.5.

The Difficulty: Cores of Cores Are Hard to Compute. There are several reasons to believe that one cannot compute kernels for the hitting set problem in constant depth using the repeated sunflower-reduction-procedure. A first idea for reaching a constant depth is to apply the reduction procedure only a constant number of times (instead of d times). Indeed, it is not immediately clear that a "core of cores" is not already a core in the first round – so do we actually need more than *one* round? Unfortunately, the answer is "yes, we do": Figure 1 shows an example where $\{a, b\}$ is a 2-core of the 2-cores, but it is not a 2-core of the original hypergraph. For a more complex example, where $d-1$ rounds are needed to arrive at a constant size kernel, consider the trees T_d^ℓ (defined in detail later on) that are perfectly balanced trees of depth d with $\ell+1$ children per node for a number $\ell \geq k$ – and now consider the hypergraph H^d that has one hyperedge for each node of T_d^ℓ and this hyperedge contains all the nodes on the path from the node to the root r. Now, for $i > 0$ we have k-cores$(H^i) = H^{i-1}$ and the latter hypergraphs all have a size of at least the arbitrarily large ℓ for $i > 1$. Thus, we need to apply the "core of cores" procedure at least $d-1$ times before arriving at a hypergraph whose size depends only on the parameter.

A second, more promising idea is the observation that it might be possible to somehow "collapse" two (and then, hopefully, all) applications of the sunflower-reduction-procedure "into a single application." Unfortunately, we also run into a problem here, namely in the "collapsed color coding process." In essence, color coding is great at ensuring that certain vertex sets are disjoint (namely those vertex sets that receive different colors), but fails at enforcing that the same vertices are used in different hyperedges – which is exactly what is needed when the definition of some H_i refers to H_{i-1}, which in turn refers to some H_{i-2}.

These problems with avoiding the build-up of additional layers with rising d have led Chen et al. [11] to the conjecture that the build-up is unavoidable and that all parallel kernelization algorithms for $p_{k,d}$-HITTING-SET have a runtime that is linear in d. We agree with Chen et al. in their assessment that the computation of the H_i presumably necessitates a linear circuit depth – but, nevertheless, we will refute their conjecture in the following.

The Solution: Pseudo-Cores As a Replacement For Cores. Our idea is *not* to compute the sets H_i (we do not see how this can be done in constant time), but to compute hypergraphs H'_i with rather similar properties (formally, they will form matryoshka sequences as well) that we *can* compute in constant time for all d and i. We introduce a new notion of *k-pseudo-cores of level i* and H'_i will be the hypergraph whose edges are the k-pseudo-cores of level i. Crucially, the definition of H'_i (only) refers directly to the original input graph H and its

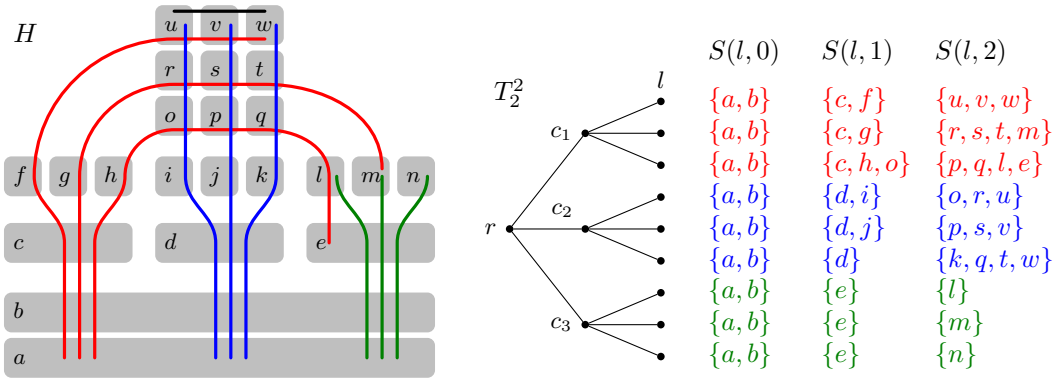


Figure 2 A T_2^2-pseudo-sunflower S for the level 2 pseudo-core $\{a,b\}$ in the hypergraph H. The four properties of pseudo-sunflowers hold: In "column $S(l,0)$" we always have the pseudo-core, the union of each row is a hyperedge, the sets in a row form a partition of this hyperedge, and – most importantly – we have the disjointness property at each "branch" of the tree. This property requires that for column $S(l,1)$ the sets of all red vertices, of all blue vertices, and of all green vertices are pairwise disjoint; whereas for column $S(l,2)$ it requires that the three red sets are pairwise disjoint, likewise for the three blue sets, and the three green sets. However, it is permissible (and the case) that a red vertex in the third column is the same as green vertex in the third or the second column.

hyperedges can be obtained from H directly using color coding. At the same time, the H'_i will form a matryoshka sequence and, hence, just as for the H_i, the core of any sunflower of H'_{i-1} must already be present in H'_i.

The definition of pseudo-cores is somewhat technical. We will, however, show that all cores are pseudo-cores of level 1, cores of cores are pseudo-cores of level 2, and so on. The reverse implication does not hold (for instance, pseudo-cores of level 2 need not be cores of cores). For a "level" L and a number k, let T_L^k denote the rooted tree in which all leafs are at the same depth L and all inner nodes have exactly $k+1$ children. The root of T_L^k will always be called r in the following. Thus, T_1^k is just a star consisting of r and its $k+1$ children, while in T_2^k each of the $k+1$ children of r has $k+1$ new children, leading to $(k+1)^2$ leafs in total. For each $l \in \text{leafs}(T_L^k) = \{\, l \mid l \text{ is a leaf of } T_L^k \,\}$ there is a unique path $(l^0, l^1, \ldots, l^L)$ from $l^0 = r$ to $l^L = l$. An example for the following definition is shown in Figure 2.

▶ **Definition 4.1** (Pseudo-Sunflowers and Pseudo-Cores). Let $H = (V,E)$ be a hypergraph and let L and k be fixed. A set $C \subseteq V$ is called a *k-pseudo-core of level L in H* if there exists a mapping $S\colon \text{leafs}(T_L^k) \times \{0,1,\ldots,L\} \to 2^V$, called a *$T_L^k$-pseudo-sunflower for H with pseudo-core C*, such that for all $l, m \in \text{leafs}(T_L^k)$ with $l \neq m$ we have:

1. $S(l,0) = C$.
2. $S(l,0) \cup S(l,1) \cup \cdots \cup S(l,L) \in E$ and let us write $S(l)$ for this hyperedge.
3. $S(l,i) \cap S(l,j) = \emptyset$ for $0 \leq i < j \leq L$, but $S(l,i) \neq \emptyset$ for $i \in \{1,\ldots,L\}$.
4. Let $z \in \{1,\ldots,L\}$ be the smallest number such that $l^z \neq m^z$, that is, z is the depth where the path from r to l and the path from r to m diverge for the first time. Then $S(l,z) \cap S(m,z) = \emptyset$ must hold.

▶ **Definition 4.2.** For a hypergraph $H = (V,E)$ and numbers k and $i \geq 1$ let $H'_i = \big(V, \{C \mid C \text{ is a } k\text{-pseudo-core of level } i \text{ of } H\}\big)$ and let $H'_0 = H$.

To get some intuition, let us have a closer look at H'_1. As the following lemma shows, pseudo-cores and cores are still *very* closely related at this first level – while for larger levels, we no longer have $H_i = H'_i$, but only $H_i \subseteq H'_i$.

▶ **Lemma 4.3.** *Let H be a hypergraph and k a number. Then $H_1 = H'_1$.*

Sketch of Proof: For $L = 1$, the properties of a pseudo-sunflower enforce exactly that the unions $S(l,0) \cup S(l,1)$ form petals of a sunflower with core $S(l,0)$. ◀

5 The Constant-Depth Kernelization

We show that hitting set kernels can be computed in constant depth in two steps:

1. We show that $(H'_0, \dots, H'_{d(H)})$ is a matryoshka sequence.
2. We show that all H'_i can be computed by a constant depth circuit whose depth is independent of both k and $d(H)$.

By the Kernel Theorem, Theorem 3.5, taken together, these two items yield the desired kernelization algorithm.

Step 1: Pseudo-Cores Form Matryoshka Sequences. Our first aim is to show the following theorem, which is an analogue of Lemma 3.4 for pseudo-cores:

▶ **Theorem 5.1.** *For every hypergraph H and number k, the sequence $(H'_0, \dots, H'_{d(H)})$ from Definition 4.2 is a matryoshka sequence for H and k.*

The proof consists of four lemmas, one for each of four properties of a matryoshka sequence:

▶ **Lemma 5.2.** $H'_0 = H$.

Proof. By definition. ◀

▶ **Lemma 5.3.** $d(H'_L) \leq d(H) - L$ *holds for all* $L \in \{0, \dots, d(H)\}$.

Proof. For every leaf l we have $S(l) = S(l,0) \dot\cup S(l,1) \dot\cup \cdots \dot\cup S(l,L)$ and all $S(l,i)$ for $i \in \{1, \dots, L\}$ are non-empty sets. This implies that $|S(l,0)| \leq |S(l)| - L \leq d(H) - L$. ◀

▶ **Lemma 5.4.** $k\text{-cores}(H'_L) \subseteq H'_{L+1}$ *holds for all* $L \in \{0, \dots, d(H)\}$.

Sketch of Proof. Proof by induction over L, where the base case is given by Lemma 4.3. For the inductive step, consider a k-core $C \in H'_L$, which is witnessed by a sunflower that consists of $k+1$ different T^k_L-pseudo-sunflowers. From these we construct a T^k_{L+1}-pseudo-sunflower with pseudo-core C and conclude $C \in H'_{L+1}$. ◀

▶ **Lemma 5.5.** *Every size-k hitting set of H is also a size-k hitting set of H'_L for all* $L \in \{0, \dots, d(H)\}$.

Sketch of Proof. Given a size-k hitting set X, say that it *hits a node n of T^k_L* if there is a leaf $l \in \text{leafs}(T^k_L)$ and a depth i with $n = l^i$ such that $X \cap (S(l,0) \cup \cdots \cup S(l,i)) \neq \emptyset$. With this definition, X trivially hits all leafs of T^k_L. By the fourth property of a pseudo-sunflower, if X hits all children of a node, it also hits the node. By structural induction we get that the root $r = l^0$ gets hit and, thus, $\emptyset \neq X \cap S(L,0) = X \cap C$. ◀

Step 2: Pseudo-Cores Can Be Computed in Constant Depth. Theorem 5.1 states that the hypergraphs H'_i form a matryoshka sequence and, thus, the Kernel Theorem tells us that the following hypergraph is a kernel for the hitting set problem:

$$K = (H'_0 \ominus H'_1) \cup (H'_1 \ominus H'_2) \cup \cdots \cup (H'_{d(H)-1} \ominus H'_{d(H)}) \cup H'_{d(H)}.$$

Of course, the whole effort that went into the definition of the H'_i and the proof of the matryoshka properties would be for nothing, if the H'_i were not easier to compute than the H_i.

This is exactly what we claim in the following theorem and prove in the rest of this paper: It is an analogue of Lemma 3.6 for pseudo-cores. The crucial difference in the formulation is that, now, we no longer get H'_{i-1} as input when we compute H'_i, but rather we compute H'_i "directly" from the original graph H.

▶ **Theorem 5.6** (Computing Pseudo-Cores in Constant Depth). *There is a* DLOGTIME*-uniform family of* AC*-circuits that*

1. *on input of the incidence matrix of a hypergraph $H = (V, E)$ and numbers k and L,*
2. *outputs the refinement matrix encoding of H'_L,*
3. *has constant depth (in particular, it is independent of $|V|$, $|E|$, $d(H)$, k, and L), and*
4. *has size $f(k, d(H)) \cdot |V|^{O(1)}|E|^{O(1)}$ where f is some computable function.*

To compute the encoding of H'_L, we can consider all candidate pseudo-cores in parallel. Thus, proving the theorem boils down to deciding for a subset $C \subseteq V$ whether there exists a T^k_L-pseudo-sunflower S of H whose pseudo-core is C. Of course, we wish to use color coding for this and our definition of pseudo-cores and pseudo-sunflowers was carefully crafted so that it includes only requirements of the form "these parts of these hyperedges must be disjoint" (and not – as is necessary for describing cores of cores – statements like "these hyperedges must *share* the vertices that form petals"). Unfortunately, while we no longer need to *ensure* that certain parts of different hyperedges are identical, we must be careful that we do not inadvertently *forbid* vertices to be the same across hyperedges when we "do not care whether they are the same":

▶ **Example 5.7.** Suppose we wish to find two disjoint hyperedges $e_1 = \{v_1, v_2, v_3\}$ and $e_2 = \{v_4, v_5, v_6\}$ in a hypergraph H plus another hyperedge $e_3 = \{x, y\}$ such that $x \notin e_1 \cup e_2$, but do not care whether $y \in e_1 \cup e_2$ holds or not. We can easily enforce the disjointness properties by coloring v_1 to v_6 using colors 1 to 6 and x using color 7. However, how should we color y for which *we do not care about disjointness* (at least with respect to e_1 and e_2)? Fixing any of the colors 1 to 3 for y or any of the colors 4 to 6 (or, for that matter, any other color) would be wrong, since this would enforce either $y \notin e_2$ or $y \notin e_1$ (or both).

Fortunately, there is a way out of the dilemma: we consider all feasible colors y could get in parallel. To formalize this "trick", we define a technical problem in which an undirected graph G is used to specify which vertices in hyperedges of a hypergraph H should be different. As is customary, a *proper coloring* of an undirected graph $G = (U, F)$ is a mapping $c\colon U \to C$ to some set C of colors with $c(u) \neq c(v)$ for all $\{u, v\} \in F$. Let us write $f[X] = \{f(x) \mid x \in X\}$ for the image of a set X under a function f. For an example instance see Figure 3.

▶ **Problem 5.8.** p_G-RESTRICTED-COLORING

Instance: *A hypergraph $H = (V, E)$ and an undirected graph $G = (U, F)$ together with a partition $U = U_1 \dot\cup \cdots \dot\cup U_m$ of U.*

Parameter: $|G|$

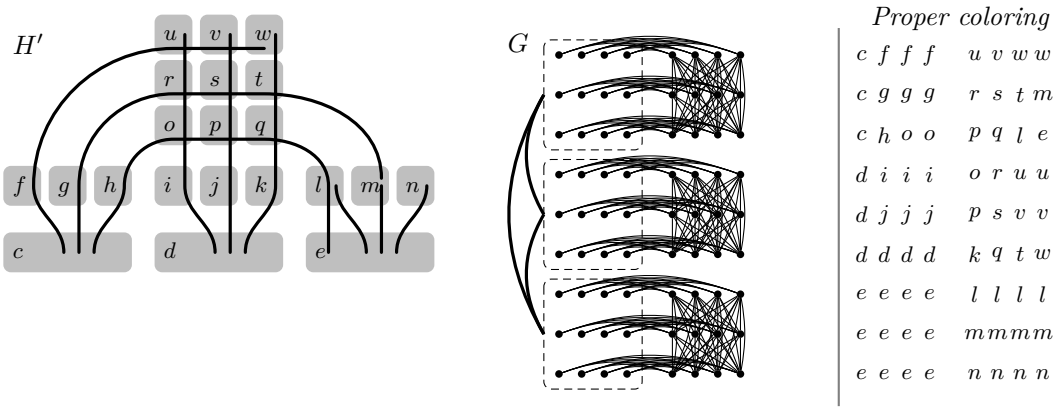


Figure 3 An instance of p_G-RESTRICTED-COLORING consisting of a hypergraph H' and a graph G (a thick edge connecting two areas with dashed borders indicates that there is an edge between each vertex of the first area and each vertex of the second area; thus, in the example, each thick edge corresponds to $12 \cdot 12 = 144$ edges). This instance is the one resulting from the reduction described in the proof of Theorem 5.6 for $L = 2$, the hypergraph H from Figure 1, and the core $\{a, b\}$ (except that we use only four vertices in G per set $S(l, i)$ instead of $d = 9$). A proper coloring is shown right (the table indicates the values $c(u) \in V(H')$ for the corresponding vertices u of G).

Question: *Is there a proper coloring $c\colon U \to V$ of G such that $c[U_i] \in E$ holds for all $i \in \{1, \dots, m\}$?*

▶ **Lemma 5.9.** *The problem p_G-RESTRICTED-COLORING can be solved by a* DLOGTIME*-uniform family of* AC*-circuits of constant depth and size $f(|G|)|V|^{O(1)}|E|^{O(1)}$ for some computable function f.*

Sketch of Proof. We show that c exists if, and only if, there is a mapping $d\colon V \to \{1, \dots, |U|\}$ with the following two properties (considering all possible proper colorings c' in parallel is the formal implementation of the above-mentioned "trick"):

1. There is a proper coloring $c'\colon U \to \{1, \dots, |U|\}$ of G such that
2. for each $i \in \{1, \dots, m\}$ there is a hyperedge $e_i \in E$ with $|d[e_i]| = |e_i|$ and $d[e_i] = c'[U_i]$.

Having proved this equivalence, we observe that we can determine c' in constant depth via "brute force", as the number of candidates depends only on the parameter. The function d can be found via color coding, since (a) the cardinality of its image depends only on the parameter and since (b) we are only interested in the values of d on the subset of V that is used by c as a color (the size of this subset depends only on the parameter). ◀

Sketch of Proof of Theorem 5.6. For each of the $|E| \cdot 2^{d(H)}$ possible pseudo-cores C we reduce the question of whether there is a T_L^k-pseudo-sunflower S with pseudo-core C to an instance for p_G-RESTRICTED-COLORING. The constructed graph G has the vertex set $\text{leafs}(T_k^L) \times \{1, \dots, L\} \times \{1, \dots, d\}$, which means that for each set $S(l, i)$ in the pseudo-sunflower's "table" there are d vertices available (which can then be mapped surjectively to the elements $S(l, i)$). We use the partition of U to ensure that $S(l)$ is always a hyperedge and we insert edges to ensure the disjointness properties of pseudo-sunflowers. ◀

Theorem 5.6 now implies Theorem 1.2 by simple standard arguments.

6 Conclusion

The results of this paper can be summarized as $p_{k,d}$-HITTING-SET $\in$ para-AC^0 or, equivalently, that kernels for the hitting set problem parameterized by k and d can be computed by a single AC^0-circuit family. This result refutes a conjecture of Chen et al. [11]. The proof introduced a new technique: Iterated applications of color coding can sometimes be "collapsed" into a single application. This collapsing is not always straightforward (as the present paper showed) and additional technical machinery may be needed to make it work.

The proof of our main result would be *much* simpler if the number of k-cores of a hypergraph depended only on the parameters k and d (since, then, only one round would be needed in the parallel algorithm). While we gave examples that refute this hope, it might be possible to tweak the idea a bit: We can compute in constant parallel time the set of all *inclusion-minimal k-cores* of a hypergraph. We believe that we can prove that the number of these inclusion-minimal k-cores depends only on k and d (unfortunately, we need rather involved and technical combinatorics and the dependence on k and d seems to be "quite bad"). Nevertheless, if this is the case, we get a different proof that $p_{k,d}$-HITTING-SET has an AC^0-kernelization, where the complexity of proving correctness is shifted away from the algorithm (which gets much simpler) towards the underlying graph theory and combinatorics (which get more complex).

References

1 Faisal N. Abu-Khzam, Michael A. Langston, Pushkar Shanbhag, and Christopher T. Symons. Scalable parallel algorithms for FPT problems. *Algorithmica*, 45(3):269–284, 2006. doi:10.1007/s00453-006-1214-1.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

3 Max Bannach, Christoph Stockhusen, and Till Tantau. Fast parallel fixed-parameter algorithms via color coding. In Thore Husfeldt and Iyad A. Kanj, editors, *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, volume 43 of *LIPIcs*, pages 224–235. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.IPEC.2015.224.

4 Max Bannach and Till Tantau. Parallel multivariate meta-theorems. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 4:1–4:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.4.

5 Max Bannach and Till Tantau. Computing hitting set kernels by AC^0-circuits. Technical Report arxiv:1801.00716 [cs.CC], ArXiv e-prints, 2018. URL: http://arxiv.org/abs/1801.00716.

6 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within nc^1. In *Proceedings: Third Annual Structure in Complexity Theory Conference, Georgetown University, Washington, D. C., USA, June 14-17, 1988*, pages 47–59. IEEE Computer Society, 1988. doi:10.1109/SCT.1988.5262.

7 Liming Cai, Jianer Chen, Rodney G. Downey, and Michael R. Fellows. Advice classes of parameterized tractability. *Ann. Pure Appl. Logic*, 84(1):119–138, 1997. doi:10.1016/S0168-0072(95)00020-8.

8 Marco Cesati and Miriam Di Ianni. Parameterized parallel complexity. In David J. Pritchard and Jeff Reeve, editors, *Euro-Par '98 Parallel Processing, 4th International Euro-*

Par Conference, Southampton, UK, September 1-4, 1998, Proceedings, volume 1470 of *Lecture Notes in Computer Science*, pages 892–896. Springer, 1998. doi:10.1007/BFb0057945.

9 Yijia Chen and Jörg Flum. Some lower bounds in parameterized ac^0. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, volume 58 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.27.

10 Yijia Chen, Jörg Flum, and Martin Grohe. Bounded nondeterminism and alternation in parameterized complexity theory. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 13–29. IEEE Computer Society, 2003. doi:10.1109/CCC.2003.1214407.

11 Yijia Chen, Jörg Flum, and Xuangui Huang. Slicewise definability in first-order logic with bounded quantifier rank. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*, volume 82 of *LIPIcs*, pages 19:1–19:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.CSL.2017.19.

12 Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3):661–701, 2015. doi:10.1007/s00453-014-9944-y.

13 P. Erdős and R. Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 1(1):85–90, 1960.

14 Jörg Flum and Martin Grohe. Describing parameterized complexity classes. In Helmut Alt and Afonso Ferreira, editors, *STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science, Antibes - Juan les Pins, France, March 14-16, 2002, Proceedings*, volume 2285 of *Lecture Notes in Computer Science*, pages 359–371. Springer, 2002. doi:10.1007/3-540-45841-7_29.

15 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

16 Rolf Niedermeier and Peter Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set. *J. Discrete Algorithms*, 1(1):89–102, 2003. doi:10.1016/S1570-8667(03)00009-1.

17 René van Bevern. Towards optimal and expressive kernelization for d-hitting set. *Algorithmica*, 70(1):129–147, 2014. doi:10.1007/s00453-013-9774-3.

Parameterized (Approximate) Defective Coloring

Rémy Belmonte[1]
University of Electro-Communications, Chofu, Tokyo, 182-8585, Japan
remy.belmonte@uec.ac.jp

Michael Lampis
Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243
LAMSADE, 75016, Paris, France
michail.lampis@dauphine.fr

Valia Mitsou
Université Paris-Diderot, IRIF, CNRS, 8243,
Université Paris-Diderot – Paris 7, 75205, Paris, France
vmitsou@irif.fr

Abstract

In Defective Coloring we are given a graph $G = (V, E)$ and two integers χ_d, Δ^* and are asked if we can partition V into χ_d color classes, so that each class induces a graph of maximum degree Δ^*. We investigate the complexity of this generalization of Coloring with respect to several well-studied graph parameters, and show that the problem is W-hard parameterized by treewidth, pathwidth, tree-depth, or feedback vertex set, if $\chi_d = 2$. As expected, this hardness can be extended to larger values of χ_d for most of these parameters, with one surprising exception: we show that the problem is FPT parameterized by feedback vertex set for any $\chi_d \neq 2$, and hence 2-coloring is the only hard case for this parameter. In addition to the above, we give an ETH-based lower bound for treewidth and pathwidth, showing that no algorithm can solve the problem in $n^{o(\mathrm{pw})}$, essentially matching the complexity of an algorithm obtained with standard techniques.

We complement these results by considering the problem's approximability and show that, with respect to Δ^*, the problem admits an algorithm which for any $\epsilon > 0$ runs in time $(\mathrm{tw}/\epsilon)^{O(\mathrm{tw})}$ and returns a solution with exactly the desired number of colors that approximates the optimal Δ^* within $(1 + \epsilon)$. We also give a $(\mathrm{tw})^{O(\mathrm{tw})}$ algorithm which achieves the desired Δ^* exactly while 2-approximating the minimum value of χ_d. We show that this is close to optimal, by establishing that no FPT algorithm can (under standard assumptions) achieve a better than 3/2-approximation to χ_d, even when an extra constant additive error is also allowed.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms, Theory of computation → Approximation algorithms analysis

Keywords and phrases Treewidth, Parameterized Complexity, Approximation, Coloring

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.10

Funding The authors were supported by the GRAPA – Graph Algorithms for Parameterized Approximation – 38593YJ PHC Sakura Project.

[1] Rémy Belmonte was supported by the ELC project (Grant-in-Aid for Scientific Research on Innovative Areas, MEXT Japan).

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 10; pp. 10:1–10:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Defective Coloring is the following problem: we are given a graph $G = (V, E)$, and two integer parameters χ_d, Δ^*, and are asked whether there exists a partition of V into at most χ_d sets (color classes), such that each set induces a graph with maximum degree at most Δ^*. Defective Coloring, which is also sometimes referred to in the literature as Improper Coloring, is a natural generalization of the classical Coloring problem, which corresponds to the case $\Delta^* = 0$. The problem was introduced more than thirty years ago [2, 17], and since then has attracted a great deal of attention [1, 4, 6, 13, 14, 16, 23, 25, 28, 32, 33, 34].

From the point of view of applications, Defective Coloring is particularly interesting in the context of wireless communication networks, where the assignment of colors to vertices often represents the assignment of frequencies to communication nodes. In many practical settings, the requirement of traditional coloring that all neighboring nodes receive distinct colors is too rigid, as a small amount of interference is often tolerable, and may lead to solutions that need drastically fewer frequencies. Defective Coloring allows one to model this tolerance through the parameter Δ^*. As a result the problem's complexity has been well-investigated in graph topologies motivated by such applications, such as unit-disk graphs and various classes of grids [5, 7, 8, 10, 26, 27]. For more background we refer to [22, 31].

In this paper we study Defective Coloring from the point of view of parameterized complexity [18, 19, 21, 39]. The problem is of course NP-hard, even for small values of χ_d, Δ^*, as it generalizes Coloring. We are therefore strongly motivated to bring to bear the powerful toolbox of structural graph parameters, such as treewidth, which have proved extremely successful in tackling other intractable hard problems. Indeed, Coloring is one of the success stories of this domain, since the complexity of this flagship problem with respect to treewidth (and related parameters pathwidth, feedback vertex set, vertex cover) is by now extremely well-understood [37, 30]. We pose the natural question of whether similar success can be achieved for Defective Coloring, or whether the addition of Δ^* significantly alters the complexity behavior of the problem. Such results are not yet known for Defective Coloring, except for the fact that it was observed in [9] that the problem admits (by standard techniques) a roughly $(\chi_d\Delta^*)^{\text{tw}}$-time algorithm, where tw is the graph's treewidth. In parameterized complexity terms, this shows that the problem is FPT parameterized by $\text{tw} + \Delta^*$. One of our main motivating questions is whether this running time can be improved qualitatively (is the problem FPT parameterized only by tw?) or quantitavely.

Our first result is to establish that the problem is W-hard not just for treewidth, but also for several much more restricted structural graph parameters, such as pathwidth, tree-depth, and feedback vertex set. We recall that for Coloring, the standard ${\chi_d}^{\text{tw}}$ algorithm is FPT by tw, as graphs of bounded treewidth also have bounded chromatic number (Lemma 1). Our result shows that the complexity of the problem changes drastically with the addition of the new parameter Δ^*, and it appears likely that tw must appear in the exponent of Δ^* in the running time, even when Δ^* is large. More strongly, we establish this hardness even for the case $\chi_d = 2$, which corresponds to the problem of partitioning a graph into two parts so as to minimize their maximum degree. This identifies Defective Coloring as another member of a family of generalizations of Coloring (such as Equitable Coloring or List Coloring) which are hard for treewidth [20].

As one might expect, the W-hardness results on Defective Coloring parameterized by treewidth (or pathwidth, or tree-depth) easily carry over for values of χ_d larger than 2. Surprisingly, we show that this is *not* the case for the parameter feedback vertex set, for which the only W-hard case is 2-coloring: we establish with a simple win/win argument that

Table 1 Summary of results. Hardness results for tree-depth imply the same bounds for treewidth and pathwidth. Conversely, algorithms which apply to treewidth apply also to all other parameters.

Parameter	Result (Exact solution)	Ref.	Result (Approximation)	Ref.
Feedback Vertex Set	W[1]-hard for $\chi_d = 2$	Thm 2	+1-approximation in time $\text{fvs}^{O(\text{fvs})}$	Cor 28
	FPT for $\chi_d \neq 2$	Thm 20		
Tree-depth	W[1]-hard for any $\chi_d \geq 2$	Thm 2	W[1]-hard to color with $(3/2-\epsilon)\chi_d + O(1)$ colors	Thm 26
Treewidth, Pathwidth	No $n^{o(\text{pw})}$ or $n^{o(\text{tw})}$ algorithm under ETH	Thm 14	$(1+\epsilon)$-approximation for Δ^* in $(\text{tw}/\epsilon)^{O(\text{tw})}$	Thm 23
			2-approximation for χ_d in $\text{tw}^{O(\text{tw})}$	Thm 25
Vertex Cover	$\text{vc}^{O(\text{vc})}$ algorithm	Thm 21		

the problem is FPT for any other value of χ_d. We also show that if one considers sufficiently restricted parameters, such as vertex cover, the problem does eventually become FPT.

Our second step is to enhance the W-hardness result mentioned above with the aim of determining as precisely as possible the complexity of Defective Coloring parameterized by treewidth. Our reduction for tree-depth and feedback vertex set is quadratic in the parameter, and hence implies that no algorithm can solve the problem in time $n^{o(\sqrt{\text{tw}})}$ under the Exponential Time Hypothesis (ETH) [29]. We therefore present a second reduction, which applies only to pathwidth and treewidth, but manages to show that no algorithm can solve the problem in time $n^{o(\text{pw})}$ or $n^{o(\text{tw})}$ under the ETH. This lower bound is tight, as it matches asymptotically the exponent given in the algorithm of [9].

To complement the above results, we also consider the problem from the point of view of (parameterized) approximation. Here things become significantly better: we give an algorithm using a technique of [36] which for any χ_d and error $\epsilon > 0$ runs in time $(tw/\epsilon)^{O(\text{tw})}n^{O(1)}$ and approximates the optimal value of Δ^* within a factor of $(1+\epsilon)$. Hence, despite the problem's W-hardness, we produce a solution arbitrarily close to optimal in FPT time.

Motivated by this algorithm we also consider the complementary approximation problem: given Δ^* find a solution that comes as close to the minimum number of colors needed as possible. By building on the approximation algorithm for Δ^*, we are able to present a $(\text{tw})^{O(\text{tw})}n^{O(1)}$ algorithm that achieves a 2-approximation for this problem. One can observe that this is not far from optimal, since an FPT algorithm with approximation ratio better than $3/2$ would contradict the problem's W-hardness for $\chi_d = 2$. However, this simple argument is unsatisfying, because it does not rule out algorithms with a ratio significantly better than $3/2$, if one also allows a small additive error; indeed, we observe that when parameterized by feedback vertex set the problem admits an FPT algorithm that approximates the optimal χ_d within an additive error of just 1. To resolve this problem we present a gap-introducing version of our reduction which, for any i produces an instance for which the optimal value of χ_d is either $2i$, or at least $3i$. In this way we show that, when parameterized by tree-depth, pathwidth, or treewidth, approximating the optimal value of χ_d better than $3/2$ is "truly" hard, and this is not an artifact of the problem's hardness for 2-coloring.

2 Definitions and Preliminaries

For a graph $G = (V, E)$ and two integers $\chi_d \geq 1$, $\Delta^* \geq 0$, we say that G admits a (χ_d, Δ^*)-coloring if one can partition V into χ_d sets such that the graph induced by each set has

maximum degree at most Δ^*. Defective Coloring is the problem of deciding, given G, χ_d, Δ^*, whether G admits a (χ_d, Δ^*)-coloring. For $\Delta^* = 0$ this corresponds to Coloring.

We assume the reader is familiar with basic notions in parameterized complexity, such as the classes FPT and W[1]. For the relevant definitions we refer to the standard textbooks [18, 19, 21, 39]. We rely on a number of well-known graph measures: treewidth [12], pathwidth, tree-depth [38], feedback vertex set, and vertex cover, denoted respectively as $\mathrm{tw}(G), \mathrm{pw}(G), \mathrm{td}(G), \mathrm{fvs}(G), \mathrm{vc}(G)$, where we drop G if it is clear from the context.

▶ **Lemma 1.** *For any graph G we have* $\mathrm{tw}(G) - 1 \leq \mathrm{fvs}(G) \leq \mathrm{vc}(G)$ *and* $\mathrm{tw}(G) \leq \mathrm{pw}(G) \leq \mathrm{td}(G) - 1 \leq \mathrm{vc}(G)$. *Furthermore, any graph G admits a* $(\mathrm{tw}(G) + 1, 0)$*-coloring, a* $(\mathrm{pw}(G) + 1, 0)$*-coloring, a* $(\mathrm{td}(G), 0)$*-coloring, and a* $(\mathrm{fvs}(G) + 2, 0)$*-coloring.*

The Exponential Time Hypothesis (ETH) states that 3-SAT on instances with n variables and m clauses cannot be solved in time $2^{o(n+m)}$ [29]. We define the k-Multi-Colored Clique problem as follows: we are given a graph $G = (V, E)$, a partition of V into k independent sets $V_1, \ldots, V_k$, such that for all $i \in \{1, \ldots, k\}$ we have $|V_i| = n$, and we are asked if G contains a k-clique. It is well-known that this problem is W[1]-hard parameterized by k, and that it does not admit any $n^{o(k)}$ algorithm, unless the ETH is false [18].

3 W-hardness for Feedback Vertex Set and Tree-depth

The main result of this section states that deciding if a graph admits a $(2, \Delta^*)$-coloring, where Δ^* is part of the input, is W[1]-hard parameterized by either fvs or td. Because of standard relations between graph parameters (Lemma 1), this implies also the same problem's W-hardness for parameters pw and tw. As might be expected, it is not hard to extend our proof to give hardness for deciding if a (χ_d, Δ^*)-coloring exists, for any constant χ_d, parameterized by tree-depth (and hence, also treewidth and pathwidth). What is perhaps more surprising is that this cannot be done in the case of feedback vertex set. Superficially, the reason we cannot extend the reduction in this case is that one of the gadgets we use in many copies in our construction has large fvs if $\chi_d > 2$. However, we give a much more convincing reason in Theorem 20 of Section 5 where we show that Defective Coloring is FPT parameterized by fvs for $\chi_d \geq 3$, and therefore, if we could extend our reduction in this case it would prove that FPT=W[1].

The main theorem of this section is stated below. We then present the reduction in Sections 3.1, 3.2, and give the Lemmata that imply Theorem 2 in Section 3.3.

▶ **Theorem 2.** *Deciding if a graph G admits a $(2, \Delta^*)$-coloring, where Δ^* is part of the input, is W[1]-hard parameterized by* $\mathrm{fvs}(G)$. *Deciding if a graph G admits a (χ_d, Δ^*)-coloring, where $\chi_d \geq 2$ is any fixed constant and Δ^* is part of the input is W[1]-hard parameterized by* $\mathrm{td}(G)$.

3.1 Basic Gadgets

Before we proceed, we present some basic gadgets that will be useful in all the reductions of this paper (Theorems 2, 14, 26). We first define a building block $\mathcal{T}(i, j)$ which is a graph that can be properly colored with i colors, but admits no $(i-1, j)$-coloring (similar constructions appears in [28]). We then use this graph to build two gadgets: the Equality Gadget and the Palette Gadget (Definitions 5 and 8). Informally, for given χ_d, Δ^*, the equality gadget allows us to express the constraint that two vertices v_1, v_2 of a graph must receive the same color in any valid (χ_d, Δ^*)-coloring. The palette gadget will be used to express the constraint

that, among three vertices v_1, v_2, v_3, there must exist two with the same color. For both gadgets we first prove formally that they express these constraints (Lemmata 6 and 9). We then show that, under certain conditions, these gadgets can be added to any graph without significantly increasing its tree-depth or feedback vertex set (Lemmata 7 and 10).

▶ **Definition 3.** Given two integers $i > 0, j \geq 0$, we define the graph $\mathcal{T}(i,j)$ recursively as follows: $T(1,j) = K_1$ for all j; for $i > 1$, $T(i,j)$ is the graph obtained by taking $(j+1)$ disjoint copies of $T(i-1,j)$ and adding to the graph a new universal vertex.

▶ **Lemma 4.** *For all $i > 0, j \geq 0$ we have: $\mathcal{T}(i,j)$ admits an $(i,0)$-coloring; $\mathcal{T}(i,j)$ does not admit an $(i-1,j)$-coloring;* $\mathrm{td}(\mathcal{T}(i,j)) = \mathrm{pw}(\mathcal{T}(i,j)) + 1 = \mathrm{tw}(\mathcal{T}(i,j)) + 1 = i$.

▶ **Definition 5.** (Equality Gadget) For $i \geq 2, j \geq 0$, we define the graph $Q(u_1, u_2, i, j)$ as follows: Q contains $ij + 1$ disjoint copies of $\mathcal{T}(i-1,j)$ as well as two vertices u_1, u_2 which are connected to all vertices except each other.

▶ **Lemma 6.** *Let $G = (V,E)$ be a graph with $v_1, v_2 \in V$ and let G' be the graph obtained from G by adding to it a copy of $Q(u_1, u_2, \chi_\mathrm{d}, \Delta^*)$ and identifying u_1 with v_1 and u_2 with v_2. Then, any $(\chi_\mathrm{d}, \Delta^*)$-coloring of G' must give the same color to v_1, v_2. Furthermore, if there exists a $(\chi_\mathrm{d}, \Delta^*)$-coloring of G that gives the same color to v_1, v_2, this coloring can be extended to a $(\chi_\mathrm{d}, \Delta^*)$-coloring of G'.*

▶ **Lemma 7.** *Let $G = (V,E)$ be a graph, $S \subseteq V$, and G' be a graph obtained from G by repeated applications of the following operation: we select two vertices $v_1, v_2 \in V$ such that $v_1 \in S$, add a new copy of $Q(u_1, u_2, \chi_\mathrm{d}, \Delta^*)$ and identify u_i with v_i, for $i \in \{1,2\}$. Then* $\mathrm{td}(G') \leq \mathrm{td}(G \setminus S) + |S| + \chi_\mathrm{d} - 1$. *Furthermore, if $\chi_\mathrm{d} = 2$ we have* $\mathrm{fvs}(G') \leq \mathrm{fvs}(G \setminus S) + |S|$.

▶ **Definition 8.** (Palette Gadget) For $i \geq 3, j \geq 0$ we define the graph $P(u_1, u_2, u_3, i, j)$ as follows: P contains $\binom{i}{2}j + 1$ copies of $\mathcal{T}(i-2,j)$, as well as three vertices u_1, u_2, u_3 which are connected to every vertex of P except each other.

▶ **Lemma 9.** *Let $G = (V,E)$ be a graph with $v_1, v_2, v_3 \in V$ and let G' be the graph obtained from G by adding to it a copy of $P(u_1, u_2, u_3, \chi_\mathrm{d}, \Delta^*)$ and identifying u_i with v_i for $i \in \{1,2,3\}$. Then, in any $(\chi_\mathrm{d}, \Delta^*)$-coloring of G' at least two of the vertices of $\{v_1, v_2, v_3\}$ must share a color. Furthermore, if there exists a $(\chi_\mathrm{d}, \Delta^*)$-coloring of G that gives the same color to two of the vertices of $\{v_1, v_2, v_3\}$, this coloring can be extended to a $(\chi_\mathrm{d}, \Delta^*)$-coloring of G'.*

▶ **Lemma 10.** *Let $G = (V,E)$ be a graph, $S \subseteq V$, and G' be a graph obtained from G by repeated applications of the following operation: we select three vertices $v_1, v_2, v_3 \in V$ such that $v_1, v_2 \in S$, add a new copy of $P(u_1, u_2, u_3, \chi_\mathrm{d}, \Delta^*)$ and identify u_i with v_i, for $i \in \{1,2,3\}$. Then* $\mathrm{td}(G') \leq \mathrm{td}(G \setminus S) + |S| + \chi_\mathrm{d} - 2$.

3.2 Construction

We are now ready to present a reduction from k-Multi-Colored Clique. In this section we describe a construction which, given an instance of this problem (G,k) as well as an integer $\chi_\mathrm{d} \geq 2$ produces an instance of Defective Coloring. Recall that we assume that in the initial instance $G = (V,E)$ is given to us partitioned into k independent sets $V_1, \ldots, V_k$, all of which have size n. We will produce a graph $H(G, k, \chi_\mathrm{d})$ and an integer Δ^* with the property that H admits a $(\chi_\mathrm{d}, \Delta^*)$-coloring if and only if G has a k-clique. In the next section we prove the correctness of the construction and give bounds on the values of $\mathrm{td}(H)$ and $\mathrm{fvs}(H)$ to establish Theorem 2.

In our new instance we set $\Delta^* = |E| - \binom{k}{2}$. Let us now describe the graph H. Since we will repeatedly use the gadgets from Definitions 5 and 8, we will use the following convention: whenever v_1, v_2 are two vertices we have already introduced to H, when we say that we add an equality gadget $Q(v_1, v_2)$, this means that we add to H a copy of $Q(u_1, u_2, \chi_d, \Delta^*)$ and then identify u_1, u_2 with v_1, v_2 respectively (similarly for palette gadgets). To ease presentation we will gradually build the graph by describing its different conceptual parts.

Palette Part. Informally, the goal of this part is to obtain two vertices (p_A, p_B) which are guaranteed to have different colors. This part contains the following:

1. Two vertices called p_A, p_B which we will call the main palette vertices.
2. For all $i \in \{1, \ldots, \Delta^*\}$, $j \in \{A, B\}$ a vertex p^i_j.
3. For all $i \in \{1, \ldots, \Delta^*\}$, $j \in \{A, B\}$ we add an equality gadget $Q(p_j, p^i_j)$.
4. An edge between p_A, p_B.
5. For all $i \in \{1, \ldots, \Delta^*\}$, $j \in \{A, B\}$ an edge from p_j to p^i_j.

Choice Part. Informally, the goal of this part is to encode a choice of a vertex in each V_i. To this end we make $2n$ choice vertices for each color class of the original instance. The selection will be encoded by counting how many of the first n of these vertices have the same color as p_A. Formally, this part contains the following:

6. For all $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, 2n\}$ the vertex c^i_j. We call these the choice vertices.
7. For all $i \in \{1, \ldots, k\}$, $j \in \{A, B\}$ the vertex g^i_j. We call these the guard vertices.
8. For all $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, 2n\}$ edges between c^i_j and the vertices g^i_A and g^i_B.
9. For all $i \in \{1, \ldots, k\}$, $j \in \{A, B\}$ we add an equality gadget $Q(p_j, g^i_j)$.
10. If $\chi_d \geq 3$, for all $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, 2n\}$ we add a palette gadget $P(p_A, p_B, c^i_j)$.

Transfer Part. Informally, the goal of this part is to transfer the choices of the previous part to the rest of the graph. For each color class of the original instance we make $(k-1)$ "low" transfer vertices, whose deficiency will equal the choice made in the previous part, and $(k-1)$ "high" transfer vertices, whose deficiency will equal the complement of the same value. Formally, this part of H contains the following:

11. For $i, j \in \{1, \ldots, k\}$, $i \neq j$ the vertex $h_{i,j}$ and the vertex $l_{i,j}$. We call these the high and low transfer vertices.
12. For $i, j \in \{1, \ldots, k\}$, $i \neq j$ and for all $l \in \{1, \ldots, n\}$ an edge from $l_{i,j}$ to c^i_l.
13. For $i, j \in \{1, \ldots, k\}$, $i \neq j$ and for all $l \in \{n+1, \ldots, 2n\}$ an edge from $h_{i,j}$ to c^i_l.
14. For all $i, j \in \{1, \ldots, k\}$, $i \neq j$ we add an equality gadget $Q(p_A, l_{i,j})$ and an equality gadget $Q(p_A, h_{i,j})$.

Edge representation. Informally, this part contains a gadget representing each edge of G. Each gadget will contain a special vertex which will be able to receive the color of p_B if and only if the corresponding edge is part of the clique. Formally, we assume that all the vertices of each V_i are numbered $\{1, \ldots, n\}$. For each edge e of G, if e connects the vertex with index i_1 from V_{j_1} with the vertex with index i_2 from V_{j_2} (assuming without loss of generality $j_1 < j_2$) we add the following vertices and edges to H:

15. Four independent sets $L^1_e, H^1_e, L^2_e, H^2_e$ with respective sizes $n - i_1$, i_1, $n - i_2$, i_2.
16. Edges connecting the vertex l_{j_1,j_2} (respectively, $h_{j_1,j_2}, l_{j_2,j_1}, h_{j_2,j_1}$) with all vertices of the set L^1_e (respectively the sets H^1_e, L^2_e, H^2_e).
17. A vertex c_e, connected to all vertices in $L^1_e \cup H^1_e \cup L^2_e \cup H^2_e$.
18. If $\chi_d \geq 3$, for each $v \in L^1_e \cup H^1_e \cup L^2_e \cup H^2_e \cup \{c_e\}$ we add a palette gadget $P(p_A, p_B, v)$.

Finally, once we have added a gadget (as described above) for each $e \in E$, we add the following structure to H in order to ensure that we have a sufficient number of edges included in our clique:

19. A vertex c_U (universal checker) connected to all c_e for $e \in E$.
20. An equality gadget $Q(p_A, c_U)$.

Budget-Setting. Our construction is now almost done, except for the fact that some crucial vertices have degree significantly lower than Δ^* (and hence are always trivially colorable). To fix this, we will effectively lower their deficiency budget by giving them some extra neighbors. Formally, we add the following:

21. For each guard vertex g^i_j, with $j \in \{A, B\}$, we construct an independent set G^i_j of size $\Delta^* - n$ and connect it to g^i_j. For each $v \in G^i_j$ we add an equality gadget $Q(p_j, v)$.
22. For each transfer vertex $l_{i,j}$ (respectively $h_{i,j}$), we construct an independent set of size $\Delta^* - n$ and connect all its vertices to $l_{i,j}$ (or respectively to $h_{i,j}$). For each vertex v of this independent set we add an equality gadget $Q(p_A, v)$.
23. For each vertex c_e we add an independent set of size Δ^* and connect all its vertices to c_e. For each vertex v of this independent set we add an equality gadget $Q(p_B, v)$.

This completes the construction of the graph H.

3.3 Correctness

To establish Theorem 2 we need to establish three properties of the graph $H(G, k, \chi_d)$ described in the preceding section: that the existence of a k-clique in G implies that H admits a (χ_d, Δ^*)-coloring; that a (χ_d, Δ^*)-coloring of H implies the existence of a k-clique in G; and that the tree-depth and feedback vertex set of G are bounded by some function of k. These are established in the Lemmata below.

▶ **Lemma 11.** *For any $\chi_d \geq 2$, if G contains a k-clique, then the graph $H(G, k, \chi_d)$ described in the previous section admits a (χ_d, Δ^*)-coloring.*

Proof. Consider a clique of size k in G that includes exactly one vertex from each V_i. We will denote this clique by a function $f : \{1, \ldots, k\} \to \{1, \ldots, n\}$, that is, we assume that the clique contains the vertex with index $f(i)$ from V_i. We produce a (χ_d, Δ^*)-coloring of H as follows: vertex p_A receives color 1, while vertex p_B receives color 2. All vertices for which we have added an equality gadget with one endpoint identified with p_A (respectively p_B) take color 1 (respectively 2). We use Lemma 6 to properly color the internal vertices of the equality gadgets.

We have still left uncolored the choice vertices c^i_j as well as the internal vertices $L^1_e, H^1_e, L^2_e, H^2_e, c_e$ of the edge gadgets. We proceed as follows: for all $i \in \{1, \ldots, k\}$ we use color 1 on the vertices c^i_l such that $l \in \{1, \ldots, f(i)\} \cup \{n+1, \ldots, 2n - f(i)\}$; we use color 2 on all remaining choice vertices. For every $e \in E$ that is contained in the clique we color all vertices of the sets $L^1_e, H^1_e, L^2_e, H^2_e$ with color 1, and c_e with color 2. For all other edges we use the opposite coloring: we color all vertices of the sets $L^1_e, H^1_e, L^2_e, H^2_e$ with color 2, and c_e with color 1. We use Lemma 9 to properly color the internal vertices of palette gadgets, since all palette gadgets that we add use either color 1 or color 2 twice in their endpoints. This completes the coloring.

To see that the coloring we described is a (χ_d, Δ^*)-coloring, first we note that by Lemmata 6,9 internal vertices of equality and palette gadgets are properly colored. Vertices p_A, p_B have exactly Δ^* neighbors with the same color; guard vertices g^i_j have exactly n neighbors

with the same color among the choice vertices, hence exactly Δ^* neighbors with the same color overall; choice vertices have at most k neighbors of the same color, and we can assume that $k < |E| - \binom{k}{2}$; the vertex c_U has exactly $\Delta^* = |E| - \binom{k}{2}$ neighbors with color 1, since the clique contains exactly $\binom{k}{2}$ edges; all internal vertices of edge gadgets have at most one neighbor of the same color. Finally, for the transfer vertices $l_{i,j}$ and $h_{i,j}$, we note that $l_{i,j}$ (respectively $h_{i,j}$) has exactly $f(i)$ (respectively $n - f(i)$) neighbors with color 1 among the choice vertices. Furthermore, when $i < j$, $l_{i,j}$ (respectively $h_{i,j}$) has $|L_e^1|$ (respectively $|H_e^1|$) neighbors with color 1 in the edge gadgets, those corresponding to the edge e that belongs in the clique between V_i and V_j. But by construction $|L_e^1| = n - f(i)$ and $|H_e^1| = f(i)$, and with similar observations for the case $j < i$ we conclude that all vertices have deficiency at most Δ^*. ◀

▶ **Lemma 12.** *For any $\chi_d \geq 2$, if the graph $H(G, k, \chi_d)$ described in the previous section admits a (χ_d, Δ^*)-coloring, then G contains a k-clique.*

▶ **Lemma 13.** *For any $\chi_d \geq 2$, the graph $H(G, k, \chi_d)$ described in the previous section has $\mathrm{td}(H) = O(k^2 + \chi_d)$. Furthermore, if $\chi_d = 2$, then $\mathrm{fvs}(H) = O(k^2)$.*

Theorem 2 now follows directly from the reduction we have described and Lemmata 11,12,13.

4 ETH-based Lower Bounds for Treewidth and Pathwidth

In this section we present a reduction which strengthens the results of Section 3 for the parameters treewidth and pathwidth. In particular, the reduction we present here establishes that, under the ETH, the known algorithm for Defective Coloring for these parameters is essentially best possible.

We use a similar presentation order as in the previous section, first giving the construction and then the Lemmata that imply the result. Where possible, we re-use the gadgets we have already presented. The main theorem of this section states the following:

▶ **Theorem 14.** *For any fixed $\chi_d \geq 2$, if there exists an algorithm which, given a graph $G = (V, E)$ and parameters χ_d, Δ^* decides if G admits a (χ_d, Δ^*)-coloring in time $n^{o(\mathrm{pw})}$, then the ETH is false.*

4.1 Basic Gadgets

We use again the equality and palette gadgets of Section 3 (Definitions 5,8). Before proceeding, let us show that adding these gadgets to the graph does not increase the pathwidth too much. For the two types of gadget Q, P, we will call the vertices $u_1, u_2(, u_3)$ the endpoints of the gadget.

▶ **Lemma 15.** *Let $G = (V, E)$ be a graph and let G' be the graph obtained from G by repeating the following operation: find a copy of $Q(u_1, u_2, \chi_d, \Delta^*)$, or $P(u_1, u_2, u_3, \chi_d, \Delta^*)$; remove all its internal vertices from the graph; and add all edges between its endpoints which are not already connected. Then $\mathrm{tw}(G) \leq \max\{\mathrm{tw}(G'), \chi_d\}$ and $\mathrm{pw}(G) \leq \mathrm{pw}(G') + \chi_d$.*

4.2 Construction

We now describe a construction which, given an instance $G = (V, E)$, k, of k-Multi-Colored Clique and a constant χ_d returns a graph $H(G, k, \chi_d)$ and an integer Δ^* such

that H admits a (χ_d, Δ^*)-coloring if and only if G has a k-clique, and the pathwidth of H is $O(k + \chi_d)$. We use m to denote $|E|$, and we set $\Delta^* = m - \binom{k}{2}$. As in Section 3 we present the construction in steps to ease presentation, and we use the same conventions regarding adding Q and P gadgets to the graph.

Palette Part. This part repeats steps 1-5 of the construction of Section 3. We recall that this creates two main palette vertices p_A, p_B (which are eventually guaranteed to have different colors).

Choice Part. In this part we construct a sequence of independent sets, arranged in what can be thought of as a $k \times 2m$ grid. The idea is that the choice we make in coloring the first independent set of every row will be propagated throughout the row. We can therefore encode k choices of a number between 1 and n, which will encode the clique.

6. For each $i \in \{1, \ldots, k\}$, for each $j \in \{1, \ldots, 2m\}$ we construct an independent set $C_{i,j}$ of size n.
7. (Backbone vertices) For each $i \in \{1, \ldots, k\}$, for each $j \in \{1, \ldots, 2m-1\}$, for each $l \in \{A.B\}$ we construct a vertex $b^l_{i,j}$. We connect $b^l_{i,j}$ to all vertices of $C_{i,j}$ and all vertices of $C_{i,j+1}$.
8. For each backbone vertex $b^l_{i,j}$ added in the previous step, for $l \in \{A, B\}$, we add an equality gadget $Q(p_l, b^l_{i,j})$.

Edge Representation. In the $k \times 2m$ grid of independent sets we have constructed we devote two columns to represent each edge of G. In the remainder we assume some numbering of the edges of E with the numbers $\{1, \ldots, m\}$, as well as a numbering of each V_i with the numbers $\{1, \ldots, n\}$. Suppose that the j-th edge of E, where $j \in \{1, \ldots, m\}$ connects the j_1-th vertex of V_{i_1} to the j_2-th vertex of V_{i_2}, where $j_1, j_2 \in \{1, \ldots, n\}$ and $i_1, i_2 \in \{1, \ldots, k\}$. We perform the following steps for each such edge.

9. We construct four independent sets $H^1_j, L^1_j, H^2_j, L^2_j$ with respective sizes $n - j_1, j_1, n - j_2, j_2$.
10. We construct four vertices $h^1_j, l^1_j, h^2_j, l^2_j$. We connect h^1_j (respectively l^1_j, h^2_j, l^2_j) with all vertices of H^1_j (respectively L^1_j, H^2_j, L^2_j).
11. We connect h^1_j to all vertices of $C_{i_1,2j-1}$, l^1_j to all vertices of $C_{i_1,2j}$, h^2_j to all vertices of $C_{i_2,2j-1}$, l^2_j to all vertices of $C_{i_2,2j}$.
12. We add equality gadgets $Q(p_A, h^1_j), Q(p_A, l^1_j), Q(p_A, h^2_j), Q(p_A, l^2_j)$.
13. We add a checker vertex c_j and connect it to all vertices of $H^1_j \cup L^1_j \cup H^2_j \cup L^2_j$.

Validation and Budget-Setting. Finally, we add a vertex that counts how many edges we have included in our clique, as well as appropriate vertices to diminish the deficiency budget of various parts of our construction.

14. We add a universal checker vertex c_U and connect it to all vertices c_j added in step 13. We add an equality gadget $Q(p_A, c_U)$.
15. For every vertex c_j added in step 13 we construct an independent set of size Δ^* and connect all its vertices to c_j. For each vertex v in this set we add an equality gadget $Q(p_B, v)$.
16. For each vertex constructed in step 10 ($h^1_j, l^1_j, h^2_j, l^2_j$), we construct an independent set of size $\Delta^* - n$ and connect it to the vertex. For each vertex v of this independent set we add an equality gadget $Q(p_A, v)$.

17. For each backbone vertex $b^l_{i,j}$, with $l \in \{A, B\}$, we construct an independent set of size $\Delta^* - n$ and connect it to $b^l_{i,j}$. For each vertex v of this independent set we add an equality gadget $Q(p_l, v)$.
18. If $\chi_d \geq 3$, for each vertex v added in steps 6-17 we add a palette gadget $P(p_A, p_B, v)$.

4.3 Correctness

▶ **Lemma 16.** *For any $\chi_d \geq 2$, if G contains a k-clique then the graph $H(G, k, \chi_d)$ described in the previous section admits a (χ_d, Δ^*)-coloring.*

▶ **Lemma 17.** *For any $\chi_d \geq 2$, if the graph $H(G, k, \chi_d)$ described in the previous section admits a (χ_d, Δ^*)-coloring, then G contains a k-clique.*

▶ **Lemma 18.** *For the graph $H(G, k, \chi_d)$ described in the previous section* $\mathrm{pw}(H) = O(k+\chi_d)$.

The proof of Theorem 14 now follows directly from Lemmata 16,17,18.

5 Exact Algorithms for Treewidth and Other Parameters

In this section we present several exact algorithms for Defective Coloring. Theorem 19 gives a treewidth-based algorithm which can be obtained using standard techniques. Essentially the same algorithm was already sketched in [9], but we give another version here for the sake of completeness and because it is a building block for the approximation algorithm of Theorem 23. Theorem 20 uses a win/win argument to show that the problem is FPT parameterized by fvs when $\chi_d \neq 2$ and therefore explains why the reduction presented in Section 3 only works for 2 colors. Theorem 21 uses a similar argument to show that the problem is FPT parameterized by vc (for any χ_d).

▶ **Theorem 19.** *There is an algorithm which, given a graph $G = (V, E)$, parameters χ_d, Δ^*, and a tree decomposition of G of width* tw*, decides if G admits a (χ_d, Δ^*)-coloring in time $(\chi_d\Delta^*)^{O(\mathrm{tw})}n^{O(1)}$.*

▶ **Theorem 20.** Defective Coloring *is FPT parameterized by* fvs *for $\chi_d \neq 2$. More precisely, there exists an algorithm which given a graph $G = (V, E)$, parameters χ_d, Δ^*, with $\chi_d \neq 2$, and a feedback vertex set of G of size* fvs*, decides if G admits a (χ_d, Δ^*)-coloring in time* $\mathrm{fvs}^{O(\mathrm{fvs})}n^{O(1)}$.

▶ **Theorem 21.** Defective Coloring *is FPT parameterized by* vc*. More precisely, there exists an algorithm which, given a graph $G = (V, E)$, parameters χ_d, Δ^*, and a vertex cover of G of size* vc*, decides if G admits a (χ_d, Δ^*)-coloring in time* $\mathrm{vc}^{O(\mathrm{vc})}n^{O(1)}$.

6 Approximation Algorithms and Lower Bounds

In this section we present two approximation algorithms which run in FPT time parameterized by treewidth. The first algorithm (Theorem 23) is an *FPT approximation scheme* which, given a desired number of colors χ_d, is able to approximate the minimum feasible value of Δ^* for this value of χ_d arbitrarily well (that is, within a factor $(1 + \epsilon)$). The second algorithm, which also runs in FPT time parameterized by treewidth, given a desired value for Δ^*, produces a solution that approximates the minimum number of colors χ_d within a factor of 2.

These results raise the question of whether it is possible to approximate χ_d as well as we can approximate Δ^*, that is, whether there exists an algorithm which comes within a

factor $(1+\epsilon)$ (rather than 2) of the optimal number of colors. As a first response, one could observe that such an algorithm probably cannot exist, because the problem is already hard when $\chi_d = 2$, and therefore an FPT algorithm with multiplicative error less than 3/2 would imply that FPT=W[1]. However, this does not satisfactorily settle the problem as it does not rule out an algorithm that achieves a much better approximation ratio, if we allow it to also have a small additive error in the number of colors. Indeed, as we observe in Corollary 28, it is possible to obtain an algorithm which runs in FPT time parameterized by feedback vertex set and has an additive error of only 1, as a consequence of the fact that the problem is FPT for $\chi_d \geq 3$. This poses the question of whether we can design an FPT algorithm parameterized by treewidth which, given a (χ_d, Δ^*)-colorable graph, produces a coloring with $\rho\chi_d + O(1)$ colors, for $\rho < 3/2$.

In the second part of this section we settle this question negatively by showing, using a recursive construction that builds on Theorem 2, that such an algorithm cannot exist. More precisely, we present a gap-introducing version of our reduction: the ratio between the number of colors needed to color Yes and No instances remains 3/2, even as the given χ_d increases. This shows that the "correct" multiplicative approximation ratio for this problem really lies somewhere between 3/2 and 2, or in other words, that there are significant barriers impeding the design of a better than 3/2 FPT approximation for χ_d, beyond the simple fact that 2-coloring is hard.

6.1 Approximation Algorithms

Our first approximation algorithm, which is an approximation scheme for the optimal value of Δ^*, relies on a method introduced in [36] (see also [3]), and a theorem of [11]. The high-level idea is the following: intuitively, the obstacle that stops us from obtaining an FPT running time with the dynamic programming algorithm of Theorem 19 is that the dynamic program is forced to store some potentially large values for each vertex. More specifically, to characterize a partial solution we need to remember not just the color of each vertex in a bag, but also how many neighbors with the same color this vertex has already seen (which is a value that can go up to Δ^*). The main trick now is to "round" these values in order to decrease the number of possible states a vertex can be found in. To do this, we select an appropriate value δ (polynomial in $\frac{\epsilon}{\log n}$), and try to replace every value that the dynamic program would calculate with the next higher integer power of $(1+\delta)$. This has the advantage of limiting the number of possible values from Δ^* to $\log_{(1+\delta)} \Delta^* \approx \frac{\log \Delta^*}{\delta}$, and this is sufficient to obtain the promised running time. The problem is now that the rounding we applied introduces an approximation error, which is initially a factor of at most $(1+\delta)$, but may increase each time we apply an arithmetic operation as part of the algorithm. To show that this error does not get out of control we show that in any bag of the tree all values stored are within a factor $(1+\delta)^h$ of the correct ones, where h is the height of the bag. We then use a theorem of Bodlaender and Hagerup [11] which states that any tree decomposition can be balanced in such a way that its height is at most $O(\log n)$, and as a result we obtain that all values are sufficiently close to being correct.

The second algorithm we present in this section (Theorem 25) uses the approximation scheme for Δ^* to obtain an FPT 2-approximation for χ_d. The idea here is that, given a (χ_d, Δ^*)-colorable graph, we first produce a $(\chi_d, (1+\epsilon)\Delta^*)$-coloring using the algorithm of Theorem 23, and then apply a procedure which uses 2 colors for each color class of this solution but manages to divide by two the number of neighbors with the same color of every vertex. This is achieved with a simple polynomial-time local search procedure.

▶ **Theorem 22.** *[11] There is a polynomial-time algorithm which, given a graph $G = (V, E)$ and a tree decomposition of G of width* tw*, produces a tree decomposition of G of width at most* $3\mathrm{tw} + 2$ *and height* $O(\log n)$*.*

▶ **Theorem 23.** *There is an algorithm which, given a graph $G = (V, E)$, parameters χ_d, Δ^*, a tree decomposition of G of width* tw*, and an error parameter $\epsilon > 0$, either returns a $(\chi_d, (1+\epsilon)\Delta^*)$-coloring of G, or correctly concludes that G does not admit a (χ_d, Δ^*)-coloring, in time* $(\mathrm{tw}/\epsilon)^{O(\mathrm{tw})} n^{O(1)}$*.*

▶ **Lemma 24.** *There exists a polynomial-time algorithm which, given a graph with maximum degree Δ, produces a two-coloring of that graph where all vertices have at most $\Delta/2$ neighbors of the same color.*

▶ **Theorem 25.** *There is an algorithm which, given a graph $G = (V, E)$, parameters χ_d, Δ^*, and a tree decomposition of G of width* tw*, either returns a $(2\chi_d, \Delta^*)$-coloring of G, or correctly concludes that G does not admit a (χ_d, Δ^*)-coloring, in time* $(\mathrm{tw})^{O(\mathrm{tw})} n^{O(1)}$*.*

6.2 Hardness of Approximation

The main result of this section is that χ_d cannot be approximated with a factor better than $3/2$ in FPT time (for parameters tree-depth, pathwidth, or treewidth), even if we allow the algorithm to also have a constant additive error. We remark that an FPT algorithm with additive error 1 is easy to obtain for feedback vertex set (Corollary 28).

▶ **Theorem 26.** *For any fixed $\chi_d > 0$, if there exists an algorithm which, given a graph $G = (V, E)$ and a $\Delta^* \geq 0$, correctly distinguishes between the case that G admits a $(2\chi_d, \Delta^*)$-coloring, and the case that G does not admit a $(3\chi_d - 1, \Delta^*)$-coloring in FPT time parameterized by* $\mathrm{td}(G)$*, then FPT=W[1].*

▶ **Corollary 27.** *For any constants $\delta_1, \delta_2 > 0$, if there exists an algorithm which, given a graph $G = (V, E)$ that admits a (χ_d, Δ^*)-coloring and parameters χ_d, Δ^*, is able to produce a $((\frac{3}{2} - \delta_1)\chi_d + \delta_2, \Delta^*)$-coloring of G in FPT time parameterized by* $\mathrm{td}(G)$*, then FPT=W[1].*

▶ **Corollary 28.** *There is an algorithm which, given a graph $G = (V, E)$, parameters χ_d, Δ^*, and a feedback vertex set of G of size* fvs*, either returns a $(\chi_d + 1, \Delta^*)$-coloring of G, or correctly concludes that G does not admit a (χ_d, Δ^*)-coloring, in time* $(\mathrm{fvs})^{O(\mathrm{fvs})} n^{O(1)}$*.*

7 Conclusions

In this paper we classified the complexity of Defective Coloring with respect to some of the most well-studied graph parameters, given essentially tight ETH-based lower bounds for pathwidth and treewidth, and explored the parameterized approximability of the problem. Though this gives a good first overview of the problem's parameterized complexity landscape, there are several questions worth investigating next. First, is it possible to make the lower bounds of Section 4 even tighter, by precisely determining the base of the exponent in the algorithm's dependence? This would presumably rely on a stronger complexity assumption such as the SETH, as in [37]. Second, can we determine the complexity of the problem with respect to other structural parameters, such as clique-width [15], modular-width [24], or neighborhood diversity [35]? For some of these parameters the existence of FPT algorithms is already ruled out by the fact that Defective Coloring is NP-hard on cographs [9], however the complexity of the problem is unknown if we also add χ_d or Δ^* as a parameter. Finally, it would be very interesting to close the gap between 2 and $3/2$ on the performance of the best treewidth-parameterized FPT approximation for χ_d.

References

1 Nirmala Achuthan, N. R. Achuthan, and M. Simanihuruk. On minimal triangle-free graphs with prescribed k-defective chromatic number. *Discrete Mathematics*, 311(13):1119–1127, 2011. doi:10.1016/j.disc.2010.08.013.

2 James A Andrews and Michael S Jacobson. On a generalization of chromatic number. *Congressus Numerantium*, 47:33–48, 1985.

3 Eric Angel, Evripidis Bampis, Bruno Escoffier, and Michael Lampis. Parameterized power vertex cover. In Pinar Heggernes, editor, *Graph-Theoretic Concepts in Computer Science - 42nd International Workshop, WG 2016, Istanbul, Turkey, June 22-24, 2016, Revised Selected Papers*, volume 9941 of *Lecture Notes in Computer Science*, pages 97–108, 2016. doi:10.1007/978-3-662-53536-3_9.

4 Patrizio Angelini, Michael A. Bekos, Felice De Luca, Walter Didimo, Michael Kaufmann, Stephen G. Kobourov, Fabrizio Montecchiani, Chrysanthi N. Raftopoulou, Vincenzo Roselli, and Antonios Symvonis. Vertex-coloring with defects. *J. Graph Algorithms Appl.*, 21(3):313–340, 2017. doi:10.7155/jgaa.00418.

5 Júlio Araújo, Jean-Claude Bermond, Frédéric Giroire, Frédéric Havet, Dorian Mazauric, and Remigiusz Modrzejewski. Weighted improper colouring. *J. Discrete Algorithms*, 16:53–66, 2012. doi:10.1016/j.jda.2012.07.001.

6 Dan Archdeacon. A note on defective colorings of graphs in surfaces. *Journal of Graph Theory*, 11(4):517–519, 1987. doi:10.1002/jgt.3190110408.

7 Claudia Archetti, Nicola Bianchessi, Alain Hertz, Adrien Colombet, and François Gagnon. Directed weighted improper coloring for cellular channel allocation. *Discrete Applied Mathematics*, 182:46–60, 2015. doi:10.1016/j.dam.2013.11.018.

8 Jørgen Bang-Jensen and Magnús M. Halldórsson. Vertex coloring edge-weighted digraphs. *Inf. Process. Lett.*, 115(10):791–796, 2015. doi:10.1016/j.ipl.2015.05.007.

9 Rémy Belmonte, Michael Lampis, and Valia Mitsou. Defective coloring on classes of perfect graphs. *CoRR*, abs/1702.08903, 2017. arXiv:1702.08903.

10 Jean-Claude Bermond, Frédéric Havet, Florian Huc, and Cláudia Linhares Sales. Improper coloring of weighted grid and hexagonal graphs. *Discrete Math., Alg. and Appl.*, 2(3):395–412, 2010. doi:10.1142/S1793830910000747.

11 Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM J. Comput.*, 27(6):1725–1746, 1998. doi:10.1137/S0097539795289859.

12 Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, 2008. doi:10.1093/comjnl/bxm037.

13 Oleg V. Borodin, Alexandr V. Kostochka, and Matthew P. Yancey. On 11-improper 22-coloring of sparse graphs. *Discrete Mathematics*, 313(22):2638–2649, 2013. doi:10.1016/j.disc.2013.07.014.

14 I. Choi and L. Esperet. Improper coloring of graphs on surfaces. *ArXiv e-prints*, 2016. arXiv:1603.02841.

15 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

16 Lenore Cowen, Wayne Goddard, and C. Esther Jesurum. Defective coloring revisited. *Journal of Graph Theory*, 24(3):205–219, 1997. doi:10.1002/(SICI)1097-0118(199703)24:3<205::AID-JGT2>3.0.CO;2-T.

17 Lenore J. Cowen, Robert Cowen, and Douglas R. Woodall. Defective colorings of graphs in surfaces: Partitions into subgraphs of bounded valency. *Journal of Graph Theory*, 10(2):187–195, 1986. doi:10.1002/jgt.3190100207.

18 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

19 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

20 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Inf. Comput.*, 209(2):143–153, 2011. doi:10.1016/j.ic.2010.11.026.

21 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

22 Marietjie Frick. A survey of (m, k)-colorings. *Annals of Discrete Mathematics*, 55:45–57, 1993.

23 Marietjie Frick and Michael A. Henning. Extremal results on defective colorings of graphs. *Discrete Mathematics*, 126(1-3):151–158, 1994. doi:10.1016/0012-365X(94)90260-7.

24 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In Gregory Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013. doi:10.1007/978-3-319-03898-8_15.

25 Wayne Goddard and Honghai Xu. Fractional, circular, and defective coloring of series-parallel graphs. *Journal of Graph Theory*, 81(2):146–153, 2016. doi:10.1002/jgt.21868.

26 Bjarki Agust Gudmundsson, Tómas Ken Magnússon, and Björn Orri Sæmundsson. Bounds and fixed-parameter algorithms for weighted improper coloring. *Electr. Notes Theor. Comput. Sci.*, 322:181–195, 2016. doi:10.1016/j.entcs.2016.03.013.

27 Frédéric Havet, Ross J. Kang, and Jean-Sébastien Sereni. Improper coloring of unit disk graphs. *Networks*, 54(3):150–164, 2009. doi:10.1002/net.20318.

28 Frédéric Havet and Jean-Sébastien Sereni. Improper choosability of graphs and maximum average degree. *Journal of Graph Theory*, 52(3):181–199, 2006. doi:10.1002/jgt.20155.

29 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

30 Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. *CoRR*, abs/1701.06985, 2017. arXiv:1701.06985.

31 Ross J. Kang. *Improper colourings of graphs*. PhD thesis, University of Oxford, UK, 2008. URL: http://ora.ox.ac.uk/objects/uuid:a93d8303-0eeb-4d01-9b77-364113b81a63.

32 Ross J. Kang and Colin McDiarmid. The t-improper chromatic number of random graphs. *Combinatorics, Probability & Computing*, 19(1):87–98, 2010. doi:10.1017/S0963548309990216.

33 Jaehoon Kim, Alexandr V. Kostochka, and Xuding Zhu. Improper coloring of sparse graphs with a given girth, I: (0, 1)-colorings of triangle-free graphs. *Eur. J. Comb.*, 42:26–48, 2014. doi:10.1016/j.ejc.2014.05.003.

34 Jaehoon Kim, Alexandr V. Kostochka, and Xuding Zhu. Improper coloring of sparse graphs with a given girth, II: constructions. *Journal of Graph Theory*, 81(4):403–413, 2016. doi:10.1002/jgt.21886.

35 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/s00453-011-9554-x.

36 Michael Lampis. Parameterized approximation schemes using graph widths. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen,*

Denmark, July 8-11, 2014, Proceedings, Part I, volume 8572 of *Lecture Notes in Computer Science*, pages 775–786. Springer, 2014. doi:10.1007/978-3-662-43948-7_64.

37 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 777–789. SIAM, 2011. doi:10.1137/1.9781611973082.61.

38 Jaroslav Nesetril and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.

39 R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and Its Applications. OUP Oxford, 2006. URL: https://books.google.fr/books?id=mAA_OkeJxhsC.

The Relation between Polynomial Calculus, Sherali-Adams, and Sum-of-Squares Proofs

Christoph Berkholz
Humboldt-Universität zu Berlin, Germany
berkholz@informatik.hu-berlin.de

Abstract

We relate different approaches for proving the unsatisfiability of a system of real polynomial equations over Boolean variables. On the one hand, there are the static proof systems Sherali-Adams and sum-of-squares (a.k.a. Lasserre), which are based on linear and semi-definite programming relaxations. On the other hand, we consider polynomial calculus, which is a dynamic algebraic proof system that models Gröbner basis computations.

Our first result is that sum-of-squares simulates polynomial calculus: any polynomial calculus refutation of degree d can be transformed into a sum-of-squares refutation of degree $2d$ and only polynomial increase in size. In contrast, our second result shows that this is not the case for Sherali-Adams: there are systems of polynomial equations that have polynomial calculus refutations of degree 3 and polynomial size, but require Sherali-Adams refutations of degree $\Omega(\sqrt{n}/\log n)$ and exponential size.

A corollary of our first result is that the proof systems Positivstellensatz and Positivstellensatz Calculus, which have been separated over non-Boolean polynomials, simulate each other in the presence of Boolean axioms.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases Proof Complexity, Polynomial Calculus, Sum-of-Squares, Sherali-Adams

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.11

Related Version A full version of the paper is available at [4], `https://eccc.weizmann.ac.il/report/2017/154/`.

Funding Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SCHW 837/5-1.

Acknowledgements Part of this work was done during the Oberwolfach workshop 1733 *Proof Complexity and Beyond* and the author acknowledges helpful discussions with Albert Atserias, Edward Hirsch, Massimo Lauria, Joanna Ochremiak, and Iddo Tzameret on Theorem 1.1. The author thanks Edward Hirsch for providing helpful comments on an earlier version the paper.

1 Introduction

The area of *proof complexity* was founded in [8] and studies the complexity of proofs for co-NP complete problems. Traditionally, one considers proof systems for proving the unsatisfiability of (or *refuting*) a propositional formula in conjunctive normal form. If one faces a proof system, there are two important questions to ask:

1. Does the system always produce proofs of polynomial size?
2. How strong is the system compared to other proof systems?

If the answer to the first question is *yes*, in which case the system is called *p-bounded*, then NP = co-NP. Therefore, it is conjectured that no proof system is p-bounded and this

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 11; pp. 11:1–11:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


has been proven for a number of weak proof systems. For the second question, one considers the notion of *polynomial simulation*: A proof system P polynomial simulates a proof system Q if for every Q-proof of size S there is a P-proof of size $poly(S)$.

Nowadays, a large part of proof complexity focuses on weak proof systems, for which the first question has already been answered negatively. One reason for this is that they often model algorithms for solving hard problems and understanding the complexity of proofs might shed light on the complexity of algorithmic approaches that implicitly or explicitly search for proofs in the underlying proof system. The (semi-)algebraic proof systems we consider in this paper also fall into this category and are used to prove the unsatisfiability of a system $\mathcal{F}$ of real polynomial equations $f_i = 0$ over n Boolean variables $x_j \in \{0, 1\}$.[1] On the one hand, we consider *polynomial calculus*, which is a dynamic algebraic proof system that allows to derive new polynomial equations that follow from $\mathcal{F}$ line-by-line. This proof system was introduced in [7] to model Gröbner basis computations and proofs of *degree* d (where the degree of all polynomials in the derivation is bounded by d) can be found in time $n^{O(d)}$ by a bounded-degree variant of the Gröbner basis algorithm.

On the other hand, we consider the semi-algebraic proof system *Sherali-Adams* and the stronger *sum-of-squares* proof system. They are based on the linear and semi-definite programming hierarchies of Sherali-Adams [22] and Lasserre [15] and can be used to prove the unsatisfiability of a system of polynomial equations and inequalities. Proofs of degree d can be found algorithmically by solving a linear program (for Sherali-Adams) or a semi-definite program (for sum-of-squares) of size $n^{O(d)}$. Contrary to polynomial calculus, both systems are static in the sense that they provide the whole proof at once.

In order to compare these semi-algebraic proof systems with polynomial calculus, we first remark that it is known that both systems cannot be simulated by polynomial calculus. A simple example is the linear equation $\sum_{i=1}^{n} x_i = n + 1$, which has a refutation of linear size and degree 2 in Sherali-Adams and sum-of-squares, but requires polynomial calculus refutations of degree $\Omega(n)$ and size $2^{\Omega(n)}$ [13]. Our first theorem states that sum-of-squares is strictly stronger than polynomial calculus.

▶ **Theorem 1.1.** *Let $\mathcal{F}$ be a system of polynomial equations over the reals. If $\mathcal{F}$ has a polynomial calculus refutation of degree d and size S, then it has a sum-of-squares refutation of degree $2d$ and size $poly(S)$.*

For the author of this paper this theorem was highly unexpected. In fact, there has been some evidence that the contrary might be true. First, in the *non-Boolean* setting there are systems of equations that are easier to refute for polynomial calculus than for sum-of-squares [12] (see Section 2.4 for a discussion). Second, even for systems of polynomial equations over Boolean variables, separations of polynomial calculus from its static version *Nullstellensatz* were known [6].

Since sum-of-squares extends Nullstellensatz, it follows that the semi-definite lifts in the sum-of-squares/Lasserre hierarchy are necessary for "flattening" a dynamic polynomial calculus proof into a static one, although polynomial calculus is a purely algebraic system without semi-definite components. Our second theorem concerns the question whether the weaker Sherali-Adams linear programming hierarchy is already able to simulate polynomial calculus. Here we have a negative answer (that we would have expected for sum-of-squares as well).

[1] Note that this subsumes the problem of refuting 3-CNF formulas, because a clause $x \vee \overline{y} \vee z$ can be encoded as polynomial equation $(1 - x)y(1 - z) = 0$.

▶ **Theorem 1.2.** *There is a system $\mathcal{F}$ of polynomial equations over $\mathbb{R}[x_1, \ldots, x_n]$ such that:*
1. *$\mathcal{F}$ has a polynomial calculus refutation of degree 3 and size $O(n^2)$.*
2. *Every Sherali-Adams refutation of $\mathcal{F}$ has degree $\Omega(\sqrt{n}/\log n)$ and size $2^{\Omega(\sqrt{n}/\log n)}$.*

The lower bound is based on a modified version of the pebbling contradictions. The original pebbling contradictions have already been used to separate Nullstellensatz degree from polynomial calculus degree [6], but it turns out that they are easy to refute in Sherali-Adams. To obtain contradictions that are hard for Sherali-Adams (and still easy for polynomial calculus), we apply a substitution trick twice: first to show that the resulting contradiction requires high degree in Sherali-Adams and second to obtain a size lower bound from a degree lower bound. We believe that both techniques are also helpful for future lower bound arguments for static proof systems.

2 Proof Systems

For this section we fix a system of real polynomial equations $\mathcal{F} = \{f_1 = 0, \ldots, f_m = 0\}$ and a system of polynomial inequalities $\mathcal{H} = \{h_1 \geqslant 0, \ldots, h_s \geqslant 0\}$ over variables $x_1, \ldots, x_n$. As it is common in propositional proof complexity, we focus on the special case of polynomial equations (and inequalities) over *Boolean* variables and consider the task of proving that a system of polynomial equations (and/or inequalities) has no 0/1-solution. To enforce Boolean variables, the axioms $x_j^2 = x_j$ are always included in the proof systems. In Section 2.4 we briefly discuss non-Boolean variants.

Algebraic proof systems are used for proving the unsatisfiability of a system of multivariate polynomial equations over some field $\mathbb{F}$. As we focus on real polynomials we set $\mathbb{F} = \mathbb{R}$, unless mentioned otherwise. Semi-algebraic proof systems are used to prove the unsatisfiability of a system of polynomial equations and/or polynomial inequalities (in this setting the polynomials are always real).

2.1 Algebraic Proof Systems: Nullstellensatz and Polynomial Calculus

Nullstellensatz [3] is a static algebraic proof system that is based on Hilbert's Nullstellensatz. A *Nullstellensatz proof* of $f = 0$ from $\mathcal{F}$ is a sequence of polynomials $(g_1, \ldots, g_m; q_1, \ldots, q_n)$ such that

$$\sum_{i=1}^{m} g_i f_i + \sum_{j=1}^{n} q_j(x_j^2 - x_j) = f. \tag{1}$$

Note that the proof is sound in the sense every 0/1-assignment that satisfies $\mathcal{F}$ also satisfies $f = 0$. The *degree* of the Nullstellensatz proof is $\max\big(\{\deg(g_i f_i) \;:\; i \in [m]\} \cup \{\deg(q_j) + 2 \;:\; j \in [n]\}\big)$. The *size* of the derivation is the sum of the sizes of the binary encoding of the polynomials f, $g_i f_i$, $q_j(x_j^2 - x_j)$, each represented as a sum of monomials. A *Nullstellensatz refutation* of $\mathcal{F}$ is a proof of $-1 = 0$ from $\mathcal{F}$, in which case $\mathcal{F}$ is *unsatisfiable* (i. e., has no 0/1-solution). The Nullstellensatz system is also complete: If $\mathcal{F}$ is an unsatisfiable system of multi-linear polynomials, then it has a refutation of degree at most n.

Nullstellensatz is a static (or one-shot) proof system, as it provides the whole proof at once. The dynamic version of Nullstellensatz is *polynomial calculus* (PC) [7]. It consists of the following derivation rules for polynomial equations $(f_i = 0) \in \mathcal{F}$, polynomials f, g, variables x_j, and numbers $a, b \in \mathbb{R}$:

$$\frac{}{f_i = 0}, \quad \frac{}{x_j^2 - x_j = 0}, \quad \frac{f = 0}{x_j f = 0}, \quad \frac{g = 0 \qquad f = 0}{ag + bf = 0}. \tag{2}$$

A *polynomial calculus derivation* of $f = 0$ from $\mathcal{F}$ is a sequence $(r_1 = 0, \ldots, r_L = 0)$ of polynomial equations that are iteratively derived using the rules (2) and lead to $f = r_L = 0$. The *degree* of a derivation is the maximum degree of the polynomials in the derivation and the *size* is the sum of the sizes of the binary encoding of the polynomials in the derivation. A *polynomial calculus refutation* is a derivation of $-1 = 0$. It is straightforward to check that polynomial calculus simulates Nullstellensatz: If $\mathcal{F}$ has a Nullstellensatz refutation of degree d and size N, then it has a polynomial calculus refutation of degree d and size polynomial in N.

In both systems proofs of bounded degree d can be found in time $n^{O(d)}$: for Nullstellensatz the coefficients of the polynomials can be computed by solving a system of linear equations of size $n^{O(d)}$, and for polynomial calculus this can be done by using a bounded degree variant of the Gröbner basis algorithm [7].

2.2 Semi-algebraic proof systems: Sherali-Adams, Sum-of-Squares, Positivstellensatz

Sherali-Adams is a static proof system that models the Sherali-Adams lift-and-project hierarchy of linear programming relaxations [22]. It can also be viewed as an extension of the Nullstellensatz system. A *Sherali-Adams proof* of $f \geqslant 0$ from $(\mathcal{F}, \mathcal{H})$ is a sequence of polynomials $(g_1, \ldots, g_m; q_1, \ldots, q_n; p_0, \ldots, p_s)$ such that

$$\sum_{i=1}^{m} g_i f_i + \sum_{j=1}^{n} q_j(x_j^2 - x_j) + p_0 + \sum_{\ell=1}^{s} p_\ell h_\ell = f,$$

and where every p_ℓ $(\ell \geqslant 0)$ has the form $p_\ell = \sum_{A,B} a_{A,B}^\ell \prod_{j \in A} x_j \prod_{j \in B}(1 - x_j)$ with non-negative coefficients $a_{A,B}^\ell$.[2] Note that the polynomials $p_\ell \colon \mathbb{R}^n \to \mathbb{R}$ are non-negative in $[0,1]^n$ and hence the proof is sound in the sense every 0/1-assignment that satisfies $\mathcal{F}$ and $\mathcal{H}$ also satisfies $f \geqslant 0$. The *degree* (sometime called *rank*) of a Sherali-Adams proof is the maximum degree of the polynomials $g_i f_i$, $q_j(x_j^2 - x_j)$, p_0, $p_\ell h_\ell$ and the *size* is the sum of the sizes of their encoding. A Sherali-Adams refutation of $(\mathcal{F}, \mathcal{H})$ is a proof of $-1 \geqslant 0$ from $(\mathcal{F}, \mathcal{H})$. Note that every Nullstellensatz refutation of $\mathcal{F}$ is a Sherali-Adams refutation of $(\mathcal{F}, \emptyset)$ by choosing $p_0 = 0$.

Sum-of-squares (SOS) is a semi-algebraic proof system that extends Nullstellensatz and Sherali-Adams. It models the Lasserre hierarchy of semi-definite programming relaxations [15], for which reason it is sometimes called *Lasserre*, and also builds on Putinar's Positivstellensatz [21]. The difference to Sherali-Adams is that the positive polynomials p_ℓ are now sums of squares. Formally, a *sum-of-squares proof* of $f \geqslant 0$ from $(\mathcal{F}, \mathcal{H})$ is a sequence of polynomials $(g_1, \ldots, g_m; q_1, \ldots, q_n; p_0, \ldots, p_s)$ such that

$$\sum_{i=1}^{m} g_i f_i + \sum_{j=1}^{n} q_j(x_j^2 - x_j) + p_0 + \sum_{\ell=1}^{s} p_\ell h_\ell = f, \tag{3}$$

and where every p_ℓ $(\ell \geqslant 0)$ has the form $p_\ell = \sum_{c=1}^{t_\ell} (p_{\ell,c})^2$ (and is encoded as such) for arbitrary polynomials $p_{\ell,c}$ (in standard monomial form). Again, the *degree* of a proof is the maximum degree of the polynomials $g_i f_i$, $q_j(x_j^2 - x_j)$, p_0, $p_\ell h_\ell$, the *size* is the sum of the sizes of their encoding. A *sum-of-squares refutation* is a proof of $-1 \geqslant 0$. It is not hard to see

[2] We assume that the p_ℓ are explicitely provided in this form, whereas g_i and q_j are arbitrary polynomials encoded in the standard way as a sum of monomials.

that the positive polynomials $p = \sum_{A,B} a_{A,B} \prod_{j\in A} x_j \prod_{j\in B}(1-x_j)$ in the Sherali-Adams proof system have a sum-of-squares proof (from $\mathcal{F} = \mathcal{H} = \emptyset$) of degree $|A| + |B| + 1$ and size $poly(p)$. It immediately follows that sum-of-squares simulates Sherali-Adams (the same argument is also implicit in [2, 16]).

▶ **Lemma 2.1.** *If $(\mathcal{F}, \mathcal{H})$ has a Sherali-Adams refutation of degree d and size N, then it has a sum-of-squares refutation of degree $d+1$ and size $poly(N)$.*

Another semi-algebraic system that is related to sum-of-squares is *Positivstellensatz*. It builds on Stengle's Positivstellensatz (independently proven by Krivine [14] and Stengle [23]), which has also been used to define a hierarchy of relaxations, see [20]. Our definition of the Positivstellensatz proof system follows the one introduced in [12], a different way of formalising Stengle's Positivstellensatz as a proof system (without focusing on complexity) was presented in [17]. We remark that Stengle's Positivstellensatz and the Positivstellensatz proof system as defined in [12] do not necessarily include the Boolean axioms $x_j^2 - x_j$ and also work for polynomials over non-Boolean variables. To be precise, we will call the system that is named "Positivstellensatz" in [12] "non-Boolean Positivstellensatz" in this paper (see Section 2.4). To define the proof system, we consider for the system of polynomial inequalities $\mathcal{H} = \{h_1 \geqslant 0, \dots, h_s \geqslant 0\}$ the system $\widehat{\mathcal{H}} = \{\prod_{\ell\in I} h_\ell \geqslant 0 \ : \ I \subseteq [s]\}$, which extends $\mathcal{H}$ by taking products of polynomial inequalities. Clearly, $(\mathcal{F}, \mathcal{H})$ is satisfiable if and only if $(\mathcal{F}, \widehat{\mathcal{H}})$ is satisfiable. A *Positivstellensatz proof* of $f \geqslant 0$ from $(\mathcal{F}, \mathcal{H})$ is a sum-of-squares proof of $f \geqslant 0$ from $(\mathcal{F}, \widehat{\mathcal{H}})$. Note that on systems of polynomial equations (where $\mathcal{H} = \emptyset$) sum-of-squares and Positivstellensatz are the same.

One way of combining polynomial calculus with semi-algebraic proof systems is as follows. Note that a Sherali-Adams, sum-of-squares, or Positivstellensatz proof of $f \geqslant 0$ can be decomposed to

$$g + p_0 + \sum_{\ell} p_\ell h_\ell = f, \tag{4}$$

$$\sum_{i=1}^{m} g_i f_i + \sum_{j=1}^{n} q_j (x_j^2 - x_j) = g, \tag{5}$$

where (5) is a Nullstellensatz proof of $g = 0$. By replacing this Nullstellensatz proof of $g = 0$ with a polynomial calculus proof of $g = 0$, we obtain dynamic versions of the static semi-algebraic proof systems. The dynamic version of Positivstellensatz is called *Positivstellensatz calculus* and was also introduced in [12]. However, the proof of Theorem 1.1 (in particular Lemma 3.1) implies that Positivstellensatz and Positivstellensatz calculus can simulate each other.

▶ **Corollary 2.2.** *If $(\mathcal{F}, \mathcal{H})$ has a Positivstellensatz calculus refutation of degree d and size S, then it has a Positivstellensatz refutation of degree $2d$ and size $poly(S)$.*

Proof. By definition, a Positivstellensatz calculus refutation of $(\mathcal{F}, \mathcal{H})$ is a polynomial calculus derivation of $-1 - p_0 - \sum_\ell p_\ell h_\ell$ from $\mathcal{F}$, where $h_\ell \in \widehat{\mathcal{H}}$. By Lemma 3.1, there is a degree-$2d$, size $poly(S)$ sum-of-squares proof of non-negativity of

$$-\big(-1 - p_0 - \textstyle\sum_\ell p_\ell h_\ell\big)^2 = -1 - 2p_0 - p_0^2 - (2 + 2p_0)\big(\sum_\ell p_\ell h_\ell\big) - \big(\sum_\ell \sum_{\ell'} p_\ell p_{\ell'} h_{\ell'} h_\ell\big), \tag{6}$$

from $(\mathcal{F}, \widehat{\mathcal{H}})$, which in turn is a Positivstellensatz refutation of $(\mathcal{F}, \mathcal{H})$. ◀

For completeness, we mention that there are also dynamic semi-algebraic proof systems that are based on the Lovász-Schrijver lift-and-project method [18] and where one can infer polynomial *inequalities* line-by-line (see [11] for an overview). These systems are, however, much stronger and somewhat different from the proof systems considered in this paper.

2.3 Twin variables

In all the proof systems mentioned above, it might be useful to introduce *twin variables*: for every variable x_j one has available the formal variable $x_j^{\neg}$ that expresses its "negation" $1 - x_j$. To ensure that they are complementary, the additional polynomial equality $x_j + x_j^{\neg} = 1$ is always present in $\mathcal{F}$. Except for Sherali-Adams this does not change the definition of the proof systems, as it only affects the input encoding. For Sherali-Adams with twin variables, it is additionally assumed that every p_ℓ has now the form $p_\ell = \sum_{A,B} a^\ell_{A,B} \prod_{j\in A} x_j \prod_{j\in B} x_j^{\neg}$ [9].

Note that inclusion of twin variables does not affect the degree of a refutation, but it might affect the size, as for example the polynomial $\prod_{j\in[n]}(1-x_j)$, which has size $2^{\Theta(n)}$, can be more succinctly expressed as $\prod_{j\in[n]} x_j^{\neg}$, which is of size $\Theta(n)$. We are, however, not aware of any formal separation of (semi-)algebraic proof systems with and without twin variables with respect to proof size.

Twin variables are particularly useful when encoding CNF formulas into polynomial equations. It is known that polynomial calculus with twin variables, which is called *polynomial calculus resolution* (PCR) [1], can polynomially simulate the resolution calculus [7, 1]. The same is true for Sherali-Adams [9] and hence sum-of-squares (by Lemma 2.1), but not for Nullstellensatz[3].

▶ Remark. Theorem 1.1 and Theorem 1.2 remain true in the presence of twin variables.

2.4 The non-Boolean case

It is also conceivable to consider (semi-)algebraic proof systems over non-Boolean variables. In this case the additional Boolean axioms $x_j^2 - x_j = 0$ are omitted in the definitions (formally, we require that $q_j = 0$ in the above definitions). Note that there is no meaningful non-Boolean variant of the Sherali-Adams proof system, as its correctness (specifically, the non-negativity of the polynomials p_ℓ) crucially depends on the fact that all variables are between 0 and 1. However, non-Boolean variants of Nullstellensatz, polynomial calculus, sum-of-squares, and Positivstellensatz are still sound proof systems. It follows from Stengle's Positivstellensatz [23], that Positivstellensatz is also refutational complete in this setting. For sum-of-squares this does only hold if we put additional requirements on $\mathcal{F} \cup \mathcal{H}$ (being *Archimedian* [21]). Non-Boolean Nullstellensatz and polynomial calculus are only complete over algebraically closed fields (such as the complex numbers).

We remark that in these systems it is no longer the case that every unsatisfiable multi-linear system of equations over n variables has a refutation of degree n: for example, the so-called telescopic system $\mathcal{F}_n^{\text{ts}} := \{yx_1 = 1,\ x_1^2 = x_2,\ x_2^2 = x_3,\ \ldots,\ x_{n-1}^2 = x_n,\ x_n = 0\}$ requires exponential refutation degree in Nullstellensatz [5] and sum-of-squares [12]. Moreover, the same example shows that the simulation of polynomial calculus by sum-of-squares (Theorem 1.1) does not hold in the non-Boolean case:

▶ **Theorem 2.3** ([12]). *Let $\mathcal{F}_n^{ts}$ be the telescopic system as defined above.*

1. *$\mathcal{F}_n^{ts}$ has a non-Boolean Nullstellensatz (hence sum-of-squares) refutation of degree $2^{O(n)}$.*
2. *$\mathcal{F}_n^{ts}$ has a non-Boolean polynomial calculus refutation of degree $O(n)$.*
3. *Every non-Boolean sum-of-squares refutation of $\mathcal{F}_n^{ts}$ has degree $2^{\Omega(n)}$.*

[3] This essentially follows from the degree lower bounds in [6] and Lemma 4.8.

3 Sum-of-Squares Simulates Polynomial Calculus

This section is dedicated to the proof of Theorem 1.1. Let us fix an unsatisfiable system of polynomial equations $\mathcal{F} = \{f_1 = 0, \dots, f_m = 0\}$. Let $(r_1 = 0, \dots, r_L = 0)$ be a polynomial calculus derivation of $r_L = 0$ from $\mathcal{F}$ of degree d and size S. Let $\mathfrak{a}$ be the minimal integer such that every non-zero coefficient c in the proof satisfies $\mathfrak{a}^{-1} \leqslant 4c^2 \leqslant \mathfrak{a}$. Hence, the largest encoding size of a coefficient is $\Theta(\log \mathfrak{a})$. Theorem 1.1 follows immediately from the following inductive lemma.

▶ **Lemma 3.1.** *There are polynomials $q_1, \dots, q_L$ and $p_1, \dots, p_L$ of size at most poly(S) such that for every $\widehat{L} \leqslant L$ there are nonnegative coefficients a_i, b_ℓ, c_ℓ that are either zero or between $\mathfrak{a}^{-\widehat{L}}$ and $\mathfrak{a}^{\widehat{L}}$, such that*

$$\sum_{i=1}^{m}(-a_i f_i)f_i + \sum_{\ell=1}^{\widehat{L}} b_\ell q_\ell (x_{j_\ell}^2 - x_{j_\ell}) + \sum_{\ell=1}^{\widehat{L}} c_\ell p_\ell^2 = -(r_{\widehat{L}})^2 \tag{7}$$

is a sum-of-squares proof of $-(r_{\widehat{L}})^2 \geqslant 0$ of degree $2d$.

Proof. First note that (7) is indeed a sum-of-squares proof of the form (3) since

$$\sum_{\ell=1}^{\widehat{L}} b_\ell q_\ell (x_{j_\ell}^2 - x_{j_\ell}) = \sum_{j=1}^{n} \Big(\sum_{\ell\colon j_\ell = j} b_\ell q_\ell \Big) (x_j^2 - x_j)$$

and $c_\ell p_\ell^2 = (\sqrt{c_\ell} p_\ell)^2$ (as we require $c_\ell \geqslant 0$). Although we shall first provide the polynomials q_ℓ and p_ℓ, we just assume that we have already done so and postpone their definition for ease of exposition. The proof is now by induction on $\widehat{L}$ and we do a case analysis on the four types of derivation rules (2). First suppose that $r_{\widehat{L}} = f_i$ is an axiom from $\mathcal{F}$. Then we can easily derive $-(r_{\widehat{L}})^2$ in sum-of-squares by defining $p_{\widehat{L}} = q_{\widehat{L}} := 0$, setting a_i to 1 and all other coefficients to 0. The case of a Boolean axiom $r_{\widehat{L}} = x_j^2 - x_j$ is also simple. We define $q_{\widehat{L}} := -(x_j^2 - x_j)$ as well as $p_{\widehat{L}} := 0$, set $b_{\widehat{L}}$ to 1 and all other coefficients to 0 in order to derive $-(r_{\widehat{L}})^2$.

Now suppose that $r_{\widehat{L}} = x_{j'} r_{L'}$ is obtained by multiplying a previously derived polynomial $r_{L'}$ (for some $L' < L$) by a variable $x_{j'}$. By induction assumption we have a sum-of-squares proof of $-(r_{L'})^2 \geqslant 0$ of degree $2d$:

$$\sum_{i=1}^{m}(-a_i f_i)f_i + \sum_{\ell=1}^{L'} b_\ell q_\ell (x_{j_\ell}^2 - x_{j_\ell}) + \sum_{\ell=1}^{L'} c_\ell p_\ell^2 = -(r_{L'})^2. \tag{8}$$

Now we want to turn this proof into a proof of $-(x_{j'} r_{L'})^2 \geqslant 0$. Of course, we could do this by just multiplying everything by $x_{j'}^2$. However, this would increase the degree of the refutation to $2d + 2$! Instead, we use the sum of squares polynomials in order to simulate the multiplication rule in polynomial calculus without increasing the degree. We define $p_{\widehat{L}} := r_{L'} - x_{j'} r_{L'}$ as well as $q_{\widehat{L}} := -2(r_{L'})^2$ and observe that

$$(p_{\widehat{L}})^2 + q_{\widehat{L}} \cdot (x_{j'}^2 - x_{j'}) = (r_{L'})^2 - 2x_{j'}(r_{L'})^2 + x_{j'}^2 (r_{L'})^2 - 2x_{j'}^2 (r_{L'})^2 + 2x_{j'}(r_{L'})^2 \tag{9}$$

$$= (r_{L'})^2 - (x_{j'} r_{L'})^2. \tag{10}$$

By adding them to (8) we derive $-(x_{j'} r_{L'})^2 \geqslant 0$ without increasing the degree. Formally, we define $j_{\widehat{L}} := j'$, set $b_{\widehat{L}} = c_{\widehat{L}} = 1$ and obtain

$$\sum_{i=1}^{m}(-a_i f_i)f_i + \sum_{\ell=1}^{\widehat{L}} b_\ell q_\ell (x_{j_\ell}^2 - x_{j_\ell}) + \sum_{\ell=1}^{\widehat{L}} c_\ell p_\ell^2 = -(x_{j'} r_{L'})^2 = -(r_{\widehat{L}})^2.$$

The remaining case is the derivation of $r_{\widehat{L}} = a \cdot r_{L'} + b \cdot r_{L''}$ for $a, b \in \mathbb{R}$ as a linear combination of two previously derived polynomials $r_{L'}$ and $r_{L''}$. By induction assumption we have

$$\sum_{i=1}^{m}(-a'_i f_i)f_i + \sum_{\ell=1}^{L'} b'_\ell q_\ell (x_{j_\ell}^2 - x_{j_\ell}) + \sum_{\ell=1}^{L'} c'_\ell p_\ell^2 = -(r_{L'})^2 \qquad \text{and} \tag{11}$$

$$\sum_{i=1}^{m}(-a''_i f_i)f_i + \sum_{\ell=1}^{L''} b''_\ell q_\ell (x_{j_\ell}^2 - x_{j_\ell}) + \sum_{\ell=1}^{L''} c''_\ell p_\ell^2 = -(r_{L''})^2. \tag{12}$$

Our goal is to devise a sum-of-squares proof of $-(r_{\widehat{L}})^2 = -a^2(r_{L'})^2 - 2ab \cdot r_{L'} r_{L''} - b^2(r_{L''})^2$. For this we define $p_{\widehat{L}} := a \cdot r_{L'} - b \cdot r_{L''}$ and $q_{\widehat{L}} := 0$. To derive $-(r_{\widehat{L}})^2$, we multiply the sum-of-squares proof (11) by $2a^2$, multiply (12) by $2b^2$, and then add both proofs together with $(p_{\widehat{L}})^2$. More precisely, we set $a_i = 2a^2 a'_i + 2b^2 a''_i$ for all $i \in [m]$; $b_\ell = 2a^2 b'_\ell + 2b^2 b''_\ell$, $c_\ell = 2a^2 c'_\ell + 2b^2 c''_\ell$ for all $\ell \leqslant \max(L', L'')$; $c_{\widehat{L}} = 1$ and set the remaining coefficients to 0. Then we obtain

$$\sum_{i=1}^{m}(-a_i f_i)f_i + \sum_{\ell=1}^{\widehat{L}} b_\ell q_\ell (x_{j_\ell}^2 - x_{j_\ell}) + \sum_{\ell=1}^{\widehat{L}} c_\ell p_\ell^2 = -2a^2(r_{L'})^2 - 2b^2(r_{L''})^2 + (p_{\widehat{L}})^2 \tag{13}$$

$$= -a^2(r_{L'})^2 - b^2(r_{L''})^2 - 2ab \cdot r_{L'} r_{L''} \tag{14}$$

$$= -(r_{\widehat{L}})^2 \tag{15}$$

By the definition of $\mathfrak{a}$, the factors $2a^2$ and $2b^2$ are bounded by $2\mathfrak{a}^{-1}$ and $\frac{1}{2}\mathfrak{a}$ from below and above. Since by induction assumption we have $\mathfrak{a}^{-\widehat{L}+1} \leqslant a'_i, b'_\ell, c'_\ell, a''_i, b''_\ell, c''_\ell \leqslant \mathfrak{a}^{\widehat{L}-1}$, it follows that $\mathfrak{a}^{-\widehat{L}} \leqslant a_i, b_\ell, c_\ell \leqslant \mathfrak{a}^{\widehat{L}}$. This concludes the proof of Lemma 3.1. ◀

Proof of Theorem 1.1. The theorem follows immediately from Lemma 3.1, since every degree-d polynomial calculus derivation of $-1 = 0$ can be transformed into a degree-$2d$ sum-of-squares proof of non-negativity of $-(-1)^2 = -1$. By the requirements in the Lemma the size of the sum-of-squares proof is $poly(S)$. ◀

4 Sherali-Adams does not Simulate Polynomial Calculus

The system of polynomial equations that separates Sherali-Adams from polynomial calculus (Theorem 1.2) is a variant of the pebbling contradictions, which are unsatisfiable propositional formulas that are based on the black pebble game. These formulas and their variants have found several applications in propositional proof complexity. For an in-depth treatment of the history and some of the applications of pebbling in proof complexity we refer the reader to the survey [19].

Let us fix some notation. In a directed graph $\mathcal{G} = (V, E)$ we let $N^-(v) = \{u \,:\, (u, v) \in E\}$ be the set of incoming and $N^+(v) = \{w \,:\, (v, w) \in E\}$ be the set of outgoing neighbours of a vertex $v \in V$. The vertex sets $S = \{v \,:\, N^-(v) = \emptyset\}$ and $T = \{v \,:\, N^+(v) = \emptyset\}$ are called the *sources* and the *sinks* of G. A *circuit* is a directed acyclic graph $\mathcal{G}$ with a unique sink t and where every non-source vertex $v \in V \setminus S$ has two incoming neighbours.

The *(black) pebble game* is a one-player game played on a circuit $\mathcal{G} = (V, E)$. The player has available a pool of P pebbles and the game proceeds by placing and removing pebbles on the vertices of $\mathcal{G}$. In each round the player can do one of the following moves:

1. place a pebble on a source vertex $s \in S$,
2. place a pebble on $w \in V \setminus S$ if there are pebbles on both vertices in $N^-(w)$, or
3. remove an arbitrary pebble.

The player wins the game when he places a pebble on the sink node t. It is obvious, that the player can always win the game with $|V|$ pebbles and the *(black) pebbling price* $\mathrm{Peb}(\mathcal{G}) \leqslant |V|$ is the minimal number P such that the player wins the black pebble game on $\mathcal{G}$ with P pebbles. For our lower bounds we consider circuits $\mathcal{G}$ with high pebbling price.

▶ **Theorem 4.1** ([10]). *For every n there is a circuit $\mathcal{G}$ with n vertices and pebbling price* $\mathrm{Peb}(\mathcal{G}) = \Omega(n/\log n)$.

The *pebbling contradiction* $\mathcal{F}_\mathcal{G}$ for a circuit $\mathcal{G} = (V, E)$ is the system of polynomial equations over Boolean variables $\{x_v \ : \ v \in V\}$ that contains the following equations:

$$x_s = 1, \qquad \text{for all } s \in S, \tag{16}$$

$$x_u x_v = x_u x_v x_w, \qquad \text{for all } w \in V \setminus S \text{ and } N^-(w) = \{u, v\}, \text{ and} \tag{17}$$

$$x_t = 0, \qquad \text{for the sink } t. \tag{18}$$

It is easy to see that this system is unsatisfiable. Moreover, we remark that $\mathcal{F}_\mathcal{G}$ is the standard encoding of the CNF pebbling contradiction, which contains clauses x_s, $\overline{x}_u \vee \overline{x}_v \vee x_w$, and $\overline{x}_t$. As this CNF can be easily refuted in resolution using unit propagation, it follows that this system is easy to refute in any proof system that simulates resolution, such as polynomial calculus, Sherali-Adams, and sum-of-squares. For later reference, the next lemma formulates this claim for polynomial calculus. The proof is deferred to the full version of the paper [4].

▶ **Lemma 4.2.** *$\mathcal{F}_\mathcal{G}$ has a polynomial calculus refutation of degree 3 and size $O(n)$ for any n-vertex circuit $\mathcal{G}$.*

In [6] it was shown that every Nullstellensatz refutation of $\mathcal{F}_\mathcal{G}$ requires degree $\mathrm{Peb}(\mathcal{G})$ and hence this system separates Nullstellensatz degree from polynomial calculus degree. However, it is not hard to construct a Nullstellensatz refutation of $\mathcal{F}_\mathcal{G}$ that has size $\mathit{poly}(n)$. Therefore, this example does *not* separate both systems with respect to proof size. Moreover, as mentioned before, this system is also easy for Sherali-Adams (with respect to size *and* degree). To prove our separation theorem between Sherali-Adams and polynomial calculus, we modify the formula a bit in order to make it hard for Sherali-Adams, while at the same time it remains easy for polynomial calculus. We do this by substituting for every variable x_v the sum of fresh variables according to the following definition.

▶ **Definition 4.3.** Let $\mathcal{F}$ be a set of polynomial equations over variables $x_1, \ldots, x_n$ and $k \geqslant 1$. The system $\mathcal{F}[+_k]$ is obtained from $\mathcal{F}$ be replacing every variable x_i in every $f \in \mathcal{F}$ by the sum $x_{i,1} + \cdots + x_{i,k}$ of k new variables and including the additional polynomial equations $x_{i,\ell} x_{i,\ell'} = 0$ for all $i \in [n]$ and $1 \leqslant \ell < \ell' \leqslant k$.

The following lemma shows that after substitution the system remains easy to refute in polynomial calculus.

▶ **Lemma 4.4.** *Let $\mathcal{F}$ be a set of polynomial equations and suppose there is a polynomial calculus refutation of $\mathcal{F}$ of degree d and size S. Then $\mathcal{F}[+_k]$ has a polynomial calculus refutation of degree d and size $O(k^d S)$.*

Proof. We obtain the new proof by substituting all variables x_i by $x_{i,1} + \cdots + x_{i,k}$ and expand the polynomials to monomial form (this increases the size by a factor of k^d). It remains to check that the substituted equations form a polynomial calculus refutation of $\mathcal{F}[+_k]$. It is clear that a former derivation of an axiom $f \in \mathcal{F}$ is now a derivation of an substituted axiom from $\mathcal{F}[+_k]$. A derivation of a Boolean axiom $x_i^2 = x_i$ translates to

$(\sum_{\ell\in[k]} x_{i,\ell})^2 = \sum_{\ell\in[k]} x_{i,\ell}$, which can be derived using the Boolean axioms $x_{i,\ell}^2 = x_{i,\ell}$ and the additional equations $x_{i,\ell}x_{i,\ell'} = 0$ (see Definition 4.3). The substituted variant of a linear combination of two previously derived polynomials f, g is just the linear combination of the substituted versions of f and g. Multiplication by a variable x_j to a polynomial in the original proof translates to multiplying by $\sum_{\ell\in[k]} x_{j,\ell}$, which can be simulated by k separate multiplications of $x_{j,1}, \dots, x_{j,k}$ and subsequent addition steps. ◀

To obtain a system of equations that is hard for Sherali-Adams and easy for polynomial calculus we apply two substitution steps to the formula $\mathcal{F}_\mathcal{G}$ for circuits from Theorem 4.1. First, we prove that every refutation of $\mathcal{F}_\mathcal{G}[+_n]$ in Sherali-Adams requires degree $d = \mathrm{Peb}(\mathcal{G})$. In the second step we show that a degree d lower bound for an arbitrary instance $\mathcal{F}$ translates to a $2^{\Omega(d)}$ size lower bound for $\mathcal{F}[+_2]$. Together we obtain that $\mathcal{F}_\mathcal{G}[+_n][+_2]$ requires high degree *and* size in Sherali-Adams. We will use a common approach for proving lower bounds in static proof systems and define a solution for the "dual" system.

▶ **Definition 4.5.** A mapping $D\colon \mathbb{R}[x_1, \dots, x_n] \to \mathbb{R}$ is a *d-evaluation* if it satisfies the following conditions.

(D1) D is linear: $D(af + bg) = aD(f) + bD(g)$ for all $f, g \in \mathbb{R}[x_1, \dots, x_n]$ and $D(1) = 1$
(D2) D is multi-linear: $D(\prod_j x_j^{d_j}) = D(\prod_j x_j)$
(D3) $D(f \cdot f_i) = 0$ for every axiom $f_i \in \mathcal{F}$ and $f \in \mathbb{R}[x_1, \dots, x_n]$ with $\deg(f) \leqslant d - \deg(f_i)$
(D4) $D\big(\prod_{j\in A} x_j \prod_{j\in B}(1 - x_j)\big) \geqslant 0$ for all $A, B \subseteq [n]$ with $|A \cup B| \leqslant d$.

It is not hard to verify that the existence of a d-evaluation implies that there is no Sherali-Adams refutation of degree d: suppose for contradiction that there is a Sherali-Adams refutation of degree d of the form

$$\sum_{i=1}^{m} g_i f_i + \sum_{j=1}^{n} q_j(x_j^2 - x_j) + p_0 = -1, \tag{19}$$

with $p_0 = \sum_{A,B} a^0_{A,B} \prod_{j\in A} x_j \prod_{j\in B}(1 - x_j)$. Now we apply D to both sides of the equation. From (D3) it follows that $D(g_i f_i) = 0$, from (D2) we obtain $D(q_j(x_j^2 - x_j)) = 0$, and from (D4) it follows that $D(p_0) \geqslant 0$. By linearity (D1) the left hand side is evaluated to something non-negative, whereas on the right-hand side we have $D(-1) = -1$.

Due to the multi-linearity (D2) the lower bound technique actually proves something stronger. The *ml-degree* of a polynomial is the degree of its multi-linearisation, i. e., the maximum number of distinct variables in a monomial. We immediately get the following lemma.

▶ **Lemma 4.6.** *If a system of multi-linear equations $\mathcal{F}$ has a d-evaluation D, then there is no Sherali-Adams refutation of $\mathcal{F}$ that has ml-degree $\leqslant d$.*

The next lemma is proven by constructing a d-evaluation.

▶ **Lemma 4.7.** *Let $\mathcal{G}$ be a circuit with n vertices. Every Sherali-Adams refutation of $\mathcal{F}_\mathcal{G}[+_k]$ requires ml-degree at least* $\min(\mathrm{Peb}(G), k/2)$.

Proof. Let $d < \min(\mathrm{Peb}(G), k/2)$ and suppose for contradiction that there is a Sherali-Adams refutation of ml-degree d. By Lemma 4.6 it suffices to define an operator D that satisfies (D1)–(D4). We start by defining D on multi-linear terms. We call a multi-linear term *inconsistent*, if it contains two distinct variables $x_{v,\ell}$ and $x_{v,\ell'}$ for some $v \in V$. If $g = \prod_{(v,\ell)\in I} x_{v,\ell}$ is an inconsistent term, we define $D(g) := 0$. Otherwise, $g = \prod_{(v,\ell)\in I} x_{v,\ell} = \prod_{u\in U} x_{u,\ell_u}$ and the value of $D(g) := \widetilde{D}(U)$ will only depend on the set $U \subseteq V$. To define the mapping

$\widetilde{D}\colon 2^V \to \mathbb{R}$, we say that $U \subseteq V$ is *reachable*, if the player has a strategy in the black pebble game with d pebbles to reach a position where exactly the vertices in U are pebbled. The mapping is now defined as follows.

$$\widetilde{D}(U) := \begin{cases} \left(\frac{1}{k}\right)^{|U|}, \text{ if } U \text{ is reachable,} \\ 0, \text{ otherwise.} \end{cases} \tag{20}$$

We extend the definition of D to all polynomials by (multi-)linearity. Note that this completes the definition of D and immediately satisfies (D1), (D2), as well as (D3) for the axioms $x_{i,\ell}x_{i,\ell'} = 0$ introduced by Definition 4.3. To verify (D4), we have to show that $D(p) \geqslant 0$ for every polynomial $p = \prod_{(v,\ell)\in I} x_{v,\ell} \prod_{(v,\ell)\in J}(1 - x_{v,\ell})$ of degree at most d. First note that if $I \cap J \neq \emptyset$, then $D(p) = 0$ since the mapping D satisfies (D2). Therefore, we may assume that p is multi-linear when multiplied out to monomial form. If $\prod_{(v,\ell)\in I} x_{v,\ell}$ is either inconsistent or it is consistent and defines a non-reachable position, then $D(p) = 0$ and we are done. Otherwise, $D(\prod_{(v,\ell)\in I} x_{v,\ell}) = k^{-|I|}$ and we get

$$D(p) = \left(\frac{1}{k}\right)^{|I|} + \sum_{\emptyset\neq K\subseteq J} (-1)^{|K|} D\left(\prod_{(v,\ell)\in K\cup I} x_{v,\ell}\right) \geqslant \left(\frac{1}{k}\right)^{|I|} \left(1 - \sum_{z=1}^{|J|} \binom{|J|}{z} \left(\frac{1}{k}\right)^z\right).$$

Because we have have $|J| \leqslant d < k/2$ it follows that

$$\sum_{z=1}^{|J|} \binom{|J|}{z} \left(\frac{1}{k}\right)^z < \sum_{z=1}^{|J|} \left(\frac{k}{2}\right)^z \left(\frac{1}{k}\right)^z < \sum_{z=1}^{\infty} 2^{-z} = 1.$$

Hence, $D(p) > 0$ and property (D4) is proven. It remains to verify (D3) for all three types of substituted axioms. For every multi-linear term g we need to check:

$$D\left(g\cdot\left(\textstyle\sum_{\ell=1}^k x_{s,\ell}\right)\right) = D(g)\,, \tag{21}$$

$$D\left(g\cdot\left(\textstyle\sum_{\ell=1}^k x_{u,\ell}\right)\left(\textstyle\sum_{\ell=1}^k x_{v,\ell}\right)\right) = D\left(g\cdot\left(\textstyle\sum_{\ell=1}^k x_{u,\ell}\right)\left(\textstyle\sum_{\ell=1}^k x_{v,\ell}\right)\left(\textstyle\sum_{\ell=1}^k x_{w,\ell}\right)\right), \tag{22}$$

$$D\left(g\cdot\left(\textstyle\sum_{\ell=1}^k x_{t,\ell}\right)\right) = 0, \tag{23}$$

where $s \in S$ is a source, $w \in V \setminus S$ with $N^-(w) = \{u, v\}$, and t is the sink. First suppose that g is either inconsistent or defines a position U that is not reachable. In both cases everything above evaluates to 0. Hence, let $g = \prod_{u\in U} x_{u,\ell_u}$ for a reachable vertex set U. In the case of (21), we have $|U| \leqslant d - 1$. If $s \in U$, then $D(x_{s,\ell_s}g) = D(g)$, since we satisfy (D2). Since the other summands $x_{s,\ell}g$ are inconsistent for $\ell \neq \ell_s$, they evaluate to 0 and the equality (21) holds. Now assume that $s \notin U$. We have that $U \cup \{s\}$ is reachable as well, since the player has at least one pebble remaining and can place it on the source s. It follows that $D\left(g \cdot \left(\sum_{\ell=1}^k x_{s,\ell}\right)\right) = k \cdot (\frac{1}{k})^{|U|+1} = (\frac{1}{k})^{|U|} = D(g)$.

Checking (22) for non-source vertices w with $N^-(w) = \{u, v\}$ is similar. Here we have $|U| \leqslant d - 3$ and by the rules of the game we know that $U \cup \{u, v\}$ is reachable if and only if $U \cup \{u, v, w\}$ is reachable. Hence, if $U \cup \{u, v\}$ is not reachable, both sides evaluate to 0. Otherwise, by a case analysis on the shape of $U \cap \{u, v, w\}$, one can easily verify that both sides evaluate to $(\frac{1}{k})^{|U|}$.

For the sink vertex t, note that since $d < \mathrm{Peb}(G)$, no position that contains t is reachable. Hence, $D(gx_{t,\ell}) = 0$ for all $\ell \in [k]$ and the equality (23) holds. This concludes the proof of the lemma. ◀

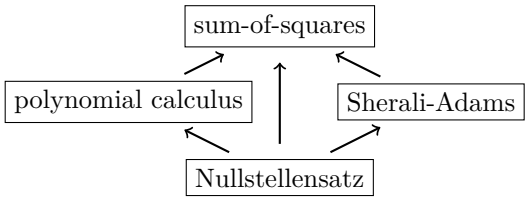


Figure 1 Relation between the proof systems. An arrow $\mathsf{P} \to \mathsf{Q}$ indicates that a proof in system P of degree d and size S can be converted into a proof in system Q of degree $O(d)$ and size $poly(S)$. Whenever there is no irreflexive arrow, it is known that the simulation does not hold.

Lemma 4.7 (together with Theorem 4.1 and Lemma 4.2) already provides a separation between degree in Sherali-Adams and polynomial calculus. To separate the proof size we need the following lifting lemma. The proof is deferred to the full version of the paper [4].

▶ **Lemma 4.8.** *Let $\mathcal{F}$ be a system of multi-linear polynomial equations and let P be one of the proof systems Nullstellensatz, Sherali-Adams, or sum-of-squares. If every P-refutation of $\mathcal{F}$ has ml-degree at least d, then every P-refutation of $\mathcal{F}[+_2]$ has ml-degree at least d and size $\Omega(2^d)$.*

By combining Lemma 4.7 and Lemma 4.8 we can now prove Theorem 1.2

Proof of Theorem 1.2. Let $\mathcal{G}$ be a circuit from Theorem 4.1 on k vertices. By Lemma 4.7 we obtain that $\mathcal{F}_{\mathcal{G}}[+_k]$ requires Sherali-Adams refutations of ml-degree $\Omega(k/\log k)$. By Lemma 4.8 it follows that every Sherali-Adams refutation of $\mathcal{F}_{\mathcal{G}}[+_k][+_2]$ requires ml-degree (and hence degree) $\Omega(k/\log k)$ and size $2^{\Omega(k/\log k)}$. On the other hand, Lemma 4.2 combined with Lemma 4.4 shows that $\mathcal{F}_{\mathcal{G}}[+_k][+_2]$ has a polynomial calculus refutation of degree 3 and size $O(k^4)$. Since $\mathcal{F}_{\mathcal{G}}[+_k][+_2]$ has $n = 2k^2$ variables, the theorem follows. ◀

5 Conclusions

We compared the static semi-algebraic proof systems Sherali-Adams and sum-of-squares with polynomial calculus, a dynamic algebraic proof system. The main results show that sum-of-squares simulates polynomial calculus (Theorem 1.1), while Sherali-Adams is not able to do so (Theorem 1.2). The relations between the proof systems considered in this paper are described in Figure 1.

One open question concerns the separation between polynomial calculus and Sherali-Adams. Note that the pebbling contradiction $\mathcal{F}_{\mathcal{G}}$ that separates polynomial calculus degree from Nullstellensatz degree is a system of polynomial equations that encodes a CNF formula. This is no longer the case for the substituted formula $\mathcal{F}_{\mathcal{G}}[+_k][+_2]$ that separates polynomial calculus from Sherali-Adams, and encoding $\mathcal{F}_{\mathcal{G}}[+_k][+_2]$ as a CNF blows up its size exponentially. It would therefore be nice to know whether there is a separating CNF. Note that such a CNF would have to be hard for resolution as well, which is not the case for the substituted variants of the pebbling contradictions (that are in conjunctive normal form) considered in the literature (see [19]).

References

1 Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Space complexity in propositional calculus. *SIAM J. Comput.*, 31(4):1184–1211, 2002. doi:10.1137/S0097539700366735.

2 Albert Atserias, Massimo Lauria, and Jakob Nordström. Narrow proofs may be maximally long. *ACM Trans. Comput. Log.*, 17(3):19:1–19:30, 2016. doi:10.1145/2898435.

3 Paul Beame, Russell Impagliazzo, Jan Krajíček, Toniann Pitassi, and Pavel Pudlák. Lower bounds on hilbert's nullstellensatz and propositional proofs. *Proceedings of the London Mathematical Society*, s3-73(1):1–26, 1996. doi:10.1112/plms/s3-73.1.1.

4 Christoph Berkholz. The relation between polynomial calculus, sherali-adams, and sum-of-squares proofs. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:154, 2017. URL: https://eccc.weizmann.ac.il/report/2017/154.

5 W. Dale Brownawell. Bounds for the degrees in the nullstellensatz. *Annals of Mathematics*, 126(3):577–591, 1987. URL: http://www.jstor.org/stable/1971361.

6 Josh Buresh-Oppenheim, Matthew Clegg, Russell Impagliazzo, and Toniann Pitassi. Homogenization and the polynomial calculus. *Computational Complexity*, 11(3-4):91–108, 2002. doi:10.1007/s00037-002-0171-6.

7 M. Clegg, J. Edmonds, and R. Impagliazzo. Using the Groebner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the 28th annual ACM symposium on Theory of computing*, pages 174–183, 1996.

8 Stephen A. Cook and Robert Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979.

9 Stefan S. Dantchev, Barnaby Martin, and Mark Nicholas Charles Rhodes. Tight rank lower bounds for the sherali-adams proof system. *Theor. Comput. Sci.*, 410(21-23):2054–2063, 2009. doi:10.1016/j.tcs.2009.01.002.

10 John R. Gilbert and Robert Endre Tarjan. Variations of a pebble game on graphs. Technical Report STAN-CS-78-661, Stanford University, 1978. Available at http://infolab.stanford.edu/TR/CS-TR-78-661.html.

11 Dima Grigoriev, Edward A. Hirsch, and Dmitrii V. Pasechnik. Complexity of semi-algebraic proofs. *Moscow Mathematical Journal*, 2(4):647–679, 2002. URL: http://www.ams.org/distribution/mmj/vol2-4-2002/abst2-4-2002.html.

12 Dima Grigoriev and Nicolai Vorobjov. Complexity of null- and positivstellensatz proofs. *Annals of Pure and Applied Logic*, 113(1–3):153–160, 2001.

13 Russell Impagliazzo, Pavel Pudlák, and Jirí Sgall. Lower bounds for the polynomial calculus and the gröbner basis algorithm. *Computational Complexity*, 8(2):127–144, 1999. doi:10.1007/s000370050024.

14 J. L. Krivine. Anneaux préordonnés. *Journal d'Analyse Mathématique*, 12(1):307–326, Dec 1964. doi:10.1007/BF02807438.

15 Jean B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001. doi:10.1137/S1052623400366802.

16 Massimo Lauria and Jakob Nordström. Tight size-degree bounds for sums-of-squares proofs. *Computational Complexity*, 26(4):911–948, 2017. doi:10.1007/s00037-017-0152-4.

17 H. Lombardi, N. Mnev, and M.-F. Roy. The positivstellensatz and small deduction rules for systems of inequalities. *Math. Nachr.*, 181:245–259, 1996.

18 László Lovász and Alexander Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM Journal on Optimization*, 1(2):166–190, 1991. doi:10.1137/0801013.

19 Jakob Nordström. Pebble games, proof complexity, and time-space trade-offs. *Logical Methods in Computer Science*, 9(3), 2013. doi:10.2168/LMCS-9(3:15)2013.

20 P. Parrilo. *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. PhD thesis, California Institute of Technology, 2000.

21 Mihai Putinar. Positive polynomials on compact semi-algebraic sets. *Indiana University Mathematics Journal*, 42(3):969–984, 1993. URL: http://www.jstor.org/stable/24897130.

22 H. D. Sherali and W. P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990.

23 Gilbert Stengle. A nullstellensatz and a positivstellensatz in semialgebraic geometry. *Mathematische Annalen*, 207(2):87–97, Jun 1974. doi:10.1007/BF01362149.

Genuine Lower Bounds for QBF Expansion

Olaf Beyersdorff
School of Computing, University of Leeds, United Kingdom
o.beyersdorff@leeds.ac.uk

Joshua Blinkhorn
School of Computing, University of Leeds, United Kingdom
scjlb@leeds.ac.uk

Abstract

We propose the first general technique for proving genuine lower bounds in expansion-based QBF proof systems. We present the technique in a framework centred on natural properties of winning strategies in the 'evaluation game' interpretation of QBF semantics. As applications, we prove an exponential proof-size lower bound for a whole class of formula families, and demonstrate the power of our approach over existing methods by providing alternative short proofs of two known hardness results. We also use our technique to deduce a result with manifest practical import: in the absence of propositional hardness, formulas separating the two major QBF expansion systems must have unbounded quantifier alternations.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases QBF, Proof Complexity, Lower-bound Techniques, Resolution

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.12

Acknowledgements We thank Meena Mahajan and Anil Shukla for helpful discussions on this work. We also thank the anonymous reviewers for useful suggestions, which helped to improve the presentation of this paper. Research was supported by grants from the John Templeton Foundation (grant no. 60842) and the EU (CORCON project).

1 Introduction

The central problem in *proof complexity* is to determine the size of the smallest proof for a given formula in a specified proof system. From its inception the field has borne tight and fruitful connections to open problems in computational complexity (separation of complexity classes [18, 14]) and first-order logic (separation of bounded arithmetic theories [32, 17]).

Proof complexity has since emerged as the natural theoretical counterpart of practical SAT solving, a subfield of automated reasoning that has enjoyed major success in recent years. Indeed, complexity of proofs and efficiency of solving are fundamentally related: the trace of a SAT solver on an unsatisfiable instance can be interpreted as a proof of falsity, whereby the correctness of each SAT solver is underpinned by a proof system. For example, the dominant paradigm in SAT, *conflict-driven clause learning* (CDCL), produces proofs in a system called *resolution* [14]. Lower bounds on resolution proof size therefore correspond to best-case running time for CDCL solvers. Consequently, there has been intense research activity focussed on proof-size lower bounds, and, in particular, *general techniques* for proof-size lower bounds in propositional logic (cf. [40, 14]).

Proof-theoretic techniques are arguably even more valuable in the increasingly challenging settings of modern solving. Consider the logic of *quantified Boolean formulas* (QBF), which extends propositional logic with existential and universal quantification. The succinct

encodings of problem instances afforded by this PSPACE-complete language [42] foster applications in diverse areas of computer science (automated planning [15, 22, 25, 39], formal verification [4, 35, 36], ontological reasoning [30], and more [20, 41, 28, 33]). Moreover, the more complex setting has spawned two distinct paradigms in solving and associated proof systems.

One approach uses QCDCL [24], the natural extension of the SAT technology, underpinned by the P+∀red family of QBF proof systems [6].[1] A second approach, implemented in the solver RAReQs [26], is through expansion of universal variables, embodied by the proof system ∀Exp+Res [27]. Research in proof complexity has revealed that these two paradigms are incomparable [27, 7] – that is, their underlying proof systems do not simulate one another.[2] This observation lead to the proposal of the more sophisticated expansion system IR-calc [7], which simulates both approaches.

It is fair to say that there is a distinct lack of general lower-bound techniques for QBF, especially for the expansion-based systems ∀Exp+Res and IR-calc. Researchers have of course attempted to lift lower bound techniques from propositional logic, but with mixed success. The celebrated size-width relations for resolution [3] are rendered ineffective in QBF resolution [10]. Prover-delayer games are only applicable to weaker tree-like proofs, both propositionally [38, 12] and in QBF [11]. Feasible interpolation [31] has been successfully transferred to QBF [9], but is tailored towards instances of a rather specific syntactic form.

Moreover, lifting techniques from SAT to QBF can be misleading, since it inevitably entails some degree of *non-genuineness* [16, 13]. The phenomenon of *genuine QBF hardness* – where lower bounds do *not* originate from the propositional level, as formalised in the oracle model of [13] – is a more suitable notion for the comparison of algorithms in quantified logic. Recent work [5] introduced a new technique for genuine QBF lower bounds in the QCDCL systems P+∀red. In this paper, we show that a semantically-grounded approach can also be employed in expansion-based systems, fostering the general techniques for genuine lower bounds that are currently missing.

Our contributions: framework, technique, and applications

We propose the *first genuine lower-bound technique* for QBF expansion. We introduce a framework built upon two semantically-grounded measures: *strategy size*, the minimum number of responses in a winning strategy; and *weight*, an extension of strategy size for unbounded prenex CNFs. Our technique encompasses three valuable theorems that express proof-size lower bounds for ∀Exp+Res and IR-calc solely in terms of these measures:

- Strategy size is an absolute proof-size lower bound in ∀Exp+Res (Theorem 7).
- Small strategy size implies short IR-calc proofs for bounded families (Theorem 9).
- Weight is an absolute proof-size lower bound in IR-calc (Theorem 22).

All three theorems are proved by counting *annotations*, a unique feature of expansion systems. Since propositional inferences preserve annotations, corollaries are invariably genuine QBF lower bounds in the formal sense of [13]. Thus, by providing an account of genuine hardness based on semantics, our technique offers valuable insight into the underlying reasons for hardness in expansion systems. Applications of our theorems represent important forward steps on at least three fronts.

---

[1] More precisely, QCDCL is underpinned by Q-Res [29], the special case of P+∀red in which P is resolution.

[2] Proof system P_1 simulates proof system P_2 whenever P_1-proofs and be transformed into P_2-proofs with at most polynomial increase in proof size.

Intuitive proofs. First, we provide short, semantically-intuitive proofs, supplanting the complicated *ad hoc* arguments that hitherto represented the state-of-the-art in QBF expansion lower bounds. Whereas the authors of [27] needed to invoke Craig's Interpolation Theorem [19] on the explicit expansion of their unbounded formulas $\mathcal{J}$,[3] we show their hardness as an immediate consequence of their manifest exponential strategy size. Similarly, the in-depth and lengthy proof of hardness in IR-calc [8] for the unbounded formulas of Kleine Büning et al. [29] is here replaced with a short argument that determines their exponential weight based on game semantics.

New hard formulas. Second, we demonstrate new exponential IR-calc proof-size lower bounds for an entire class of formula families. Using the product constructions of [5, 13], we combine a group of Π_2 CNFs F_i with a minimally unsatisfiable CNF ϕ. Provided the F_i have non-trivial strategy size (a natural stipulation), the strategy size of the product formula grows exponentially with the size of ϕ. We present product formulas with a Σ_3 prefix, but the method easily generalises to arbitrarily many quantifier alternations.

Bounded vs unbounded separations. Third, by applying our second theorem to bounded families in general, we prove that, in the absence of propositional hardness, any separation of the two expansion systems is unbounded. Given that IR-calc simulates Q-Res, this result has a remarkable corollary: any genuine separation of Q-Res from ∀Exp+Res is due to an unbounded formula family.

Organisation. We begin with preliminaries in Section 2 followed by the necessary background for QBF expansion in Section 3. We present our lower-bound technique for bounded CNFs and the associated applications in Section 4, and the extension to the unbounded case follows in Section 5. We offer conclusions in Section 6.

2 Preliminaries

Quantified Boolean formulas. A *literal* is a Boolean variable or its negation, a *clause* is a disjunction of literals, and a *CNF* is a conjunction of clauses. Throughout, we refer to a clause as a set of literals and to a CNF as a set of clauses.

A *quantified Boolean formula* (QBF) in *prenex conjunctive normal form* (PCNF) is denoted $F := \mathcal{Q} \cdot \phi$, where (a) $\mathcal{Q} := \mathcal{Q}_1 Z_1 \cdots \mathcal{Q}_n Z_n$ is the *quantifier prefix*, in which the Z_i are pairwise disjoint sets of Boolean variables called *blocks*, $\mathcal{Q}_i \in \{\exists, \forall\}$ for each $i \in [n]$, and $\mathcal{Q}_i \neq \mathcal{Q}_{i+1}$ for each $i \in [n-1]$, and (b) the *matrix* ϕ is a CNF over $\mathrm{vars}(F) := \bigcup_{i=1}^{n}(E_i \cup U_i)$. A PCNF is *$k$-bounded* if it has at most k universal blocks.

We denote the existential (resp. universal) variables of F by $\mathrm{vars}_\exists(F)$ (resp. $\mathrm{vars}_\forall(F)$). For a literal l, we write $\mathrm{var}(l) := z$ if $l = z$ or $l = \neg z$. For a clause C, we write $\mathrm{vars}(C) := \{\mathrm{var}(l) : l \in C\}$, and denote the set of existential (resp. universal) literals in C by $C_\exists$ (resp. $C_\forall$). The prefix $\mathcal{Q}$ imposes a linear ordering $<_F$ on the variables of F, such that $z_i <_F z_j$ holds whenever $i < j$, in which case we say that z_j *is right of* z_i and z_i *is left of* z_j. We extend $<_F$ to the blocks of F in the natural way.

A (partial) assignment ρ to the variables of F is represented as a set of literals, typically denoted $\{l_1, \ldots, l_k\}$, where literal z (resp. $\neg z$) represents the assignment $z \mapsto 1$ (resp. $z \mapsto 0$).

[3] This is our notation; in [27], the formulas are referred to simply as "(2)".

The CNF $\phi[\rho]$ is obtained from ϕ by removing any clause containing a literal in ρ, and removing the negated literals $\neg l_1, \ldots, \neg l_k$ from the remaining clauses. The *restriction of* F *by* ρ is $F[\rho] := \mathcal{Q}[\rho] \cdot \phi[\rho]$, where $\mathcal{Q}[\rho]$ is obtained from $\mathcal{Q}$ by removing the variables of ρ along with any quantifier whose associated block is rendered empty. For assignments to single variables we may omit the braces; for example, we write $F[l]$ for $F[\{l\}]$.

QBF semantics. Semantics for PCNFs are neatly described by the *two-player evaluation game*. Over the course of a game, the variables of a PCNF are assigned 0/1 values in the order of the prefix, with the $\exists$-player ($\forall$-player) choosing the values for the existential (universal) variables. When the game concludes, the players have constructed a total assignment ρ to the variables. The $\forall$-player wins if and only if ρ falsifies some clause of the matrix.

A $\forall$-*strategy* dictates how the $\forall$-player should respond to every possible move of the $\exists$-player. A $\forall$-*strategy* S for a PCNF F is a mapping from total assignments to $\mathrm{vars}_\exists(F)$ into total assignments to $\mathrm{vars}_\forall(F)$, such that, for each $i \in [n]$, $S(\alpha)$ and $S(\alpha')$ agree on the first i universal blocks whenever α and α' agree on the first i existential blocks. A strategy S is *winning* if and only if, for each α in the domain of S, $\phi[\alpha \cup S(\alpha)]$ contains the empty clause. We use the terms 'winning $\forall$-strategy' and 'countermodel' interchangeably. A PCNF is false if and only if it has a countermodel.

QBF proof systems. A refutational PCNF *proof system* (or *calculus*) P employs a set of axioms and inference rules to prove the falsity of PCNFs. A P derivation of a clause C_m from the *input PCNF* F is a sequence of clauses $\pi := C_1, \ldots, C_m$ in which (a) each C_i is either an axiom, or is derivable from previous clauses using an inference rule, and (b) C_m is the unique clause that is not the antecedent of an inference. The subderivation of C_i in π is the subsequence terminating at C_i containing only those clauses used in the derivation of C_i. The size $|\pi|$ of a derivation is the total number of literals appearing in it. A refutation is a derivation of the empty clause.

In this paper, we consider PCNF proof systems based on *resolution*. Resolution is a well-studied refutational proof system for propositional CNF formulas with a single inference rule: the *resolvent* $C_1 \cup C_2$ may be derived from clauses $C_1 \cup \{x\}$ and $C_2 \cup \{\neg x\}$ (variable x is the *pivot*). Resolution is *refutationally* sound and complete: that is, the empty clause can be derived from a CNF iff it is unsatisfiable. There exist a host of resolution-based QBF proof systems – see [8] for a detailed account.

For two PCNF proof systems P_1 and P_2, a PCNF family $\mathcal{F}$ *separates* P_1 from P_2 if $\mathcal{F}$ has polynomial-size refutations in P_1 but not in P_2. P_1 *p-simulates* P_2 if each P_2-proof can be transformed in polynomial time into a P_1-proof of the same formula [18].

3 Fundamentals of expansion-based calculi

In this section, we recall the definitions of $\forall$Exp+Res [27] and IR-calc [7] and discuss the underlying concepts of the calculi, including their use of annotations. We also cover proof restrictions and strategy extraction, both of which are central to the following discourse.

Intuition and definition. To explain the concept of expansion, we consider the example PCNF $\exists x \forall u \exists t.\, \phi(x, u, t)$. The formula is semantically equivalent to $\exists x \exists t^0 \exists t^1.\, \phi(x, 0, t^0) \wedge \phi(x, 1, t^1)$, in which the universal variable u has been 'expanded out', yielding a fully existentially quantified formula. Note that variable x, which is left of u, remains unchanged, while we have to create two duplicate copies t^0 and t^1 for the variable t, which is right of u.

To keep track of why we created these copies of t, we annotate them with the reason for their creation, i.e., we write $t^{\neg u}$ instead of t^0 (where $\neg u$ corresponds to the assignment $u \mapsto 0$) and likewise t^u instead of t^1. Syntactically, $t^{\neg u}$ and t^u are just new, distinct existential variables.

Since a single expansion doubles the formula size in the worst case, the complete expansion of a PCNF can blow up exponentially. In the worst case, an existential in the scope of n universals will require 2^n duplicate copies. Keeping track of all these duplicates requires annotations that are assignments to *sets* of the preceding universal variables.

In the basic theoretical model ∀Exp+Res [27], each axiom clause is immediately annotated with a *fixed, complete* assignment to the universal variables. The proof then proceeds exactly as a propositional resolution proof, with clauses in fully annotated variables. In short, ∀Exp+Res is propositional resolution on the conjuncts of a PCNF's complete expansion.

IR-calc, defined in [7], improves on this approach by working instead with *partial* assignments. In addition to resolution, the calculus is equipped with an *instantiation* rule by which partial annotations are grown throughout the course of the proof. To facilitate instantiation, the $\circ$ operator describes how partial assignments are combined. Formally, for each PCNF F, we define $\mathrm{ann}(F)$ to be the set of partial assignments to $\mathrm{vars}_\forall(F)$. Then for each $\tau, \sigma \in \mathrm{ann}(F)$, we define $\tau \circ \sigma := \tau \cup \{l \in \sigma \mid \neg l \notin \tau\}$.

The rules of both systems are given in Figure 1. Note that we write annotations as literal strings (e.g. $u_1 \neg u_3 \neg u_6 u_7$) rather than as sets.

Restrictions. This paper makes frequent use of restrictions of PCNFs and IR-calc refutations, operations that derive from their counterparts in propositional logic. Let π be an IR-calc refutation of a PCNF F.

As we will see, the purpose of restricting π by an assignment ρ is to obtain a refutation of the restricted formula $F[\rho]$. Naturally, one applies the assignment to the refutation and simplifies the result, eliminating all satisfied clauses in the process. The procedure differs depending on the quantification of the assigned variable.

For an *existential literal* l, the restricted refutation $\pi[l]$ is obtained as follows. First, remove all clauses containing a literal of the form l^τ for some $\tau \in \mathrm{ann}(F)$, and from the remaining clauses remove all literals of the form $\neg l^\tau$ for some $\tau \in \mathrm{ann}(F)$. Then $\pi[l]$ is the subderivation of the first occurrence of the empty clause in the resulting sequence.[4]

For a *universal literal* l that is *unopposed* in π (meaning that $\neg l$ does not appear in the annotations), the restricted derivation $\pi[l]$ is obtained from π simply by removing l from the annotations. We need only define restriction for unopposed universal literals.

Finally, for restriction by a partial assignment $\rho := \{l_1, \ldots, l_n\}$ with $\mathrm{var}(l_i)$ left of $\mathrm{var}(l_{i+1})$ for each $i \in [n-1]$, we define $\pi[\rho] := \pi_n$, where $\pi_0 := \pi$ and $\pi_i := \pi_{i-1}[l_i]$ for each $i \in [n]$, provided that each intermediate restriction is defined.

Restrictions of IR-calc refutations are central to strategy extraction, which rests upon the following two propositions. The first implies that first block universal literals are always unopposed. The second states that IR-calc refutations are closed under restrictions.

▶ **Proposition 1.** *Let π be an* IR-calc *derivation from a PCNF F whose leftmost block U is universal. There exists a function f such that, for each clause C in π,* (a) *for each annotation τ in C, the projection of τ to U is $f(C)$, and* (b) *$f(C') \subseteq f(C)$ for each C' in the subderivation of C.*

[4] That such a clause and its subderivation remain is proved as part of Proposition 2. We note that this subderivation may include weakening steps – the addition of arbitrary literals to a clause – but such steps are easily erased from a refutation.

$$\frac{}{\{l^{\tau(l)} \mid l \in C_\exists\}}\ [\text{axiom}(C,\tau)]$$

- C is a clause in the matrix of F
- τ is a total assignment to $\text{vars}_\forall(F)$ falsifying $C_\forall$
- $\tau(l)$ is the projection of τ to the universal variables left of $\text{var}(l)$

$$\frac{C_1 \cup \{x^\tau\} \qquad C_2 \cup \{\neg x^\tau\}}{C_1 \cup C_2}\ [\text{res}(C_1, C_2, x^\tau)]$$

$$\frac{}{\{l^{\tau(l)} \mid l \in C_\exists\}}\ [\text{axiom}(C)]$$

- C is a clause in the matrix of F.
- τ is the smallest assignment falsifying $C_\forall$
- $\tau(l)$ is the projection of τ to the universal variables left of $\text{var}(l)$

$$\frac{C}{\{l^{\sigma\circ\tau(l)} \mid l^\sigma \in C\}}\ [\text{inst}(C,\tau)]$$

- τ is a partial assignment to $\text{vars}_\forall(F)$.
- $\tau(l)$ is the projection of τ to the universal variables left of $\text{var}(l)$.

$$\frac{C_1 \cup \{x^\tau\} \qquad C_2 \cup \{\neg x^\tau\}}{C_1 \cup C_2}\ [\text{res}(C_1, C_2, x^\tau)]$$

Figure 1 The rules of ∀Exp+Res [27] (top) and IR-calc [7] (bottom). Note that $F = \mathcal{Q} \cdot \phi$ is the input PCNF.

▶ **Proposition 2** ([7]). *Let π be an* IR-calc *refutation of a PCNF F and let l be a literal with* $\text{var}(l) \in \text{vars}(F)$. *Then $\pi[l]$ is an* IR-calc *refutation of $F[l]$ if* (a) *l is existential, or* (b) *l is universal and unopposed in π.*

Strategy extraction. Strategy extraction is a prevalent paradigm in QBF proof complexity [23, 6, 1, 37], and has already been studied for IR-calc [7]. In summary, there exists an algorithm that takes a refutation and returns a countermodel (the *extracted strategy*).

Starting with an IR-calc refutation π of a PCNF $F := \exists X_1 \forall U_1 \cdots \exists X_n \forall U_n \exists X_{n+1} \cdot \phi$, we build a winning ∀-strategy S, viewing F as a game of n rounds. In round one, the ∃-player chooses some total assignment α_1 to X_1, and we collect the ∀-player's response β_1 simply by negating the U_1 literals appearing in the annotations of $\pi[\alpha_1]$. By Proposition 1, all such literals are unopposed, so β_1 is indeed an assignment. Any absent variables are assigned to 0, extending β_1 to a total assignment to U_1. By Proposition 2, $\pi[\alpha_1 \cup \beta_1]$ is a refutation of $\exists X_2 \forall U_2 \cdots \exists X_n \forall U_n \exists X_{n+1} \cdot \phi[\alpha_1 \cup \beta_1]$, i.e. of $F[\alpha_1 \cup \beta_1]$, so we repeat the process to obtain the ∀-player's response for the next round.

In this way, one obtains a full response $S(\alpha)$ to each total assignment α to the existentials, such that $\alpha \cup S(\alpha)$ falsifies ϕ. Moreover, $S(\alpha)$ and $S(\alpha')$ must agree up to block U_i if α and α' agree up to block X_i. This serves as a proof sketch for the following proposition.

▶ **Proposition 3** ([7]). *If π is an* IR-calc *refutation of a PCNF F, then the extracted strategy for π is a winning $\forall$-strategy for F.*

4 A technique for bounded formula families

In this section, we present our results for bounded PCNF families, culminating in a theorem with obvious practical relevance: in the absence of propositional hardness, separation of IR-calc from $\forall$Exp+Res is due to an unbounded formula family. We employ the following (unbounded) formulas from [27] (equation (2) in Section 6) as a running example.

▶ **Definition 4** ([27]). Let $\mathcal{J}$ be the PCNF family defined by $\mathcal{J}(n) := \mathcal{Q}_{\mathcal{J}}(n) \cdot \phi_{\mathcal{J}}(n)$, where

$$\begin{array}{rcl} \mathcal{Q}_{\mathcal{J}}(n) & := & \mathcal{Q}_1 \cdots \mathcal{Q}_n\,, \quad \text{where } \mathcal{Q}_i := \exists x_i \forall u_i \exists t_{2i-1} t_{2i} \text{ for each } i \in [n]\,, \\ \phi_{\mathcal{J}}(n) & := & \{(\neg t_1, \ldots, \neg t_{2n})\} \cup \bigcup_{i=1}^{n} \{(\neg x_i, t_{2i-1}), (\neg u_i, t_{2i-1}), (x_i, t_{2i}), (u_i, t_{2i})\}\,. \end{array}$$

The authors of [27] showed that this PCNF family separates IR-calc from $\forall$Exp+Res.[5] In light of our results, the fact that $\mathcal{J}$ is an unbounded family is not coincidental; indeed, we show that coercing $\mathcal{J}$ into a bounded family by variable reordering yields a PCNF family that is hard even for IR-calc.

4.1 IR-calc lower bounds by strategy size

Our principal insight for bounded families is that proof-size lower-bounds can be obtained by appealing to a natural and semantically-grounded measure we call *strategy size*. The strategy size of a false PCNF is the minimum number of responses in a winning strategy for the $\forall$-player. We recall that a winning strategy, or countermodel, is represented formally as a function (cf. Section 2) whose range is the set of responses. Strategy size is therefore defined as the minimum cardinality of the range of a countermodel.

▶ **Definition 5** (strategy size). The *strategy size* of a false QBF F is the minimum cardinality of the range of a countermodel for F. The strategy size of a PCNF family $\mathcal{F}$ is the function $\nabla_{\mathcal{F}} : \mathbb{N} \to \mathbb{N}$ mapping n to the strategy size of $\mathcal{F}(n)$.

▶ **Example 6.** For each $n \in \mathbb{N}$, the strategy size of $\mathcal{J}(n)$ is 2^n, so the strategy size of $\mathcal{J}$ is $\nabla_{\mathcal{J}}(n) = 2^n$. To see this, observe that the only way for the $\forall$-player to win the evaluation game by force is to set u_i not equal to x_i for each $i \in [n]$. This necessitates at least 2^n distinct responses. On the other hand, the range of a countermodel for $\mathcal{J}(n)$ is at most 2^n, since there are exactly n universal variables.

Now, recall that $\forall$Exp+Res works by applying propositional resolution to the clauses in the complete universal expansion of a PCNF. In fact, the conjuncts of the full expansion are exactly the allowable axiom clauses. An interesting question arises: how many such clauses must be introduced as axioms? It is perhaps not too difficult to see that the smallest unsatisfiable subset of the allowable axioms has cardinality not less than strategy size – this holds because the initial instantiations, one per axiom, encompass a complete set of responses for a winning strategy. Hence strategy size is an absolute proof-size lower-bound in $\forall$Exp+Res.

[5] In fact, the authors separated Q-Res from $\forall$Exp+Res; since IR-calc p-simulates Q-Res [7], the result stated in the text is an immediate corollary.

▶ **Theorem 7.** *A PCNF family $\mathcal{F}$ requires* ∀Exp+Res *refutations of size* $\nabla_{\mathcal{F}}(n)$.

The hardness of $\mathcal{J}$ in ∀Exp+Res is an immediate corollary to Theorem 7. Moreover, the fact that $\mathcal{J}$ has short IR-calc refutations implies that Theorem 7 does not lift to IR-calc. As we will see, the crux of this counterexample is that $\mathcal{J}$ is unbounded. We can in fact use strategy size as the basis for a lower-bound technique in IR-calc if we restrict our attention to bounded families. We introduce a technique based on counting annotations in the refutation. (Refutation size is clearly greater than the number of distinct annotations.) Of particular importance is the *final annotation*, the annotation to the final pivot.

▶ **Definition 8.** Let π be an IR-calc refutation, and let x^τ be the unique Boolean variable for which the empty clause is derived in π by resolution over the pivot variable x^τ. Then τ is the *final annotation* of π.

Now, if we dig into the details of the strategy extraction paradigm, we unearth a useful corollary to Proposition 1 from Section 3: Given a refutation of a PCNF whose first block U is universal, *all* the U-literals appearing in the annotations of π occur in the final annotation. This fact is crucial in the proof of the following theorem.

▶ **Theorem 9.** *A k-bounded PCNF family $\mathcal{F}$ requires* IR-calc *refutations of size* $\sqrt[k]{\nabla_{\mathcal{F}}(n)}$.

Proof sketch. Let $\mathcal{F}$ be a k-bounded PCNF family. We apply the pigeonhole principle multiplicatively to deduce the following: for any countermodel S for $\mathcal{F}(n)$, there exists some $i \in [k]$ for which the assignments to the i^{th} universal block U_i number at least $\lfloor \sqrt[k]{\nabla_{\mathcal{F}}(n)} \rfloor$. By Proposition 1 and the definition of strategy extraction, each such partial response appears as the projection to U_i of the final annotation of $\pi[\alpha]$, extended by zeros to a total assignment to U_i, for some existential assignment α. By the definition of restriction, each such final annotation is in fact the projection to U_i of an annotation in the original refutation. It follows that π contains at least $\sqrt[k]{\nabla_{\mathcal{F}}(n)}$ distinct annotations. ◀

We illustrate the effectiveness of Theorem 9 by proving that natural Σ_3 versions of $\mathcal{J}$ are hard even in IR-calc. We transform $\mathcal{J}$ into a bounded family $\mathcal{J}'$ by reordering the quantifier prefix, while preserving the strategy size.

▶ **Definition 10.** Let $\mathcal{J}'$ be the PCNF family defined by $\mathcal{J}'(n) := \mathcal{Q}_{\mathcal{J}'}(n) \cdot \phi_{\mathcal{J}}(n)$, where $\mathcal{Q}_{\mathcal{J}'}(n) := \exists x_1 \cdots x_n \forall u_1 \cdots u_n \exists t_1 \cdots t_{2n}$.

To see that exponential strategy size is preserved in $\mathcal{J}'$, observe that the *unique* winning strategy for the ∀-player is to play u_i not equal to x_i for each $i \in [n]$. Since $\mathcal{J}'$ is a 1-bounded PCNF family with $\nabla_{\mathcal{J}'}(n) = 2^n$, Theorem 9 yields an exponential proof-size lower bound.

▶ **Theorem 11.** *The PCNF family $\mathcal{J}'$ requires exponential-size* IR-calc *refutations.*

4.2 A new class of bounded hard families

Applying Theorem 7, we present a blueprint for a PCNF family with large strategy size, yielding a whole class of bounded families that are hard for IR-calc. For any CNF ϕ and clause C, let us write $\phi \otimes C := \{C' \cup C : C' \in \phi\}$ for the CNF obtained by augmenting each clause in ϕ with the literals of C. Consider the following construction, which is inspired by the random QBFs in [5].

▶ **Definition 12.** Let $k : \mathbb{N} \to \mathbb{N}$ be a function satisfying $k(n) = n^{\Omega(1)}$. Further, for each $n \in \mathbb{N}$, let $\{C_1^n, \ldots, C_{k(n)}^n\}$ be a minimally unsatisfiable CNF over variables T^n, and, for

$i \in [k(n)]$, let $\exists X_i^n \forall U_i^n \cdot \phi_i^n$ be variable-disjoint false PCNFs with strategy size greater than 1. Then the PCNF family $\mathcal{P}$ defined by $\mathcal{P}(n) := \mathcal{Q}_\mathcal{P}(n) \cdot \phi_\mathcal{P}(n)$ is a *linear product*, where

$$\mathcal{Q}_\mathcal{P}(n) := \exists X_1^n \cdots X_{k(n)}^n \forall U_1^n \cdots U_{k(n)}^n \exists T^n\,, \quad \text{and} \quad \phi_\mathcal{P}(n) := \bigcup_{i=1}^{k(n)} (\phi_i^n \otimes C_i^n)\,.$$

The intuition behind the construction of a linear product is this: to win the evaluation game on $\mathcal{P}(n)$, the $\forall$-player must win each 'subgame' $\exists X_i^n \forall U_i^n \cdot \phi_i^n$ in order to leave each clause C_i^n on the board. The non-trivial strategy size of the subgames causes the overall strategy size to blow up exponentially.

▶ **Lemma 13.** *Let $\mathcal{P}$ be a linear product. Then $\nabla_\mathcal{P}(n) = \exp(n^{\Omega(1)})$.*

Proof sketch. Let $\mathcal{P}$ be defined as in Definition 12. The only winning approach for the $\forall$-player – to reduce each CNF $\phi_i^n \otimes C_i^n$ to the clause C_i^n – encompasses winning strategies for each PCNF $F_i^n := \exists X_i^n \forall U_i^n \cdot \phi_i^n$. Since the F_i^n are pairwise variable disjoint, and therefore semantically independent from one another, one may deduce that the strategy size of $\mathcal{P}(n)$ is at least the product of the individual strategy sizes of the F_i^n. Hence the strategy size of $\mathcal{P}(n)$ is at least $2^{k(n)}$. It follows that $\nabla_\mathcal{P}(n) = \exp(n^{\Omega(1)})$. ◀

Since a linear product is 1-bounded, applying Theorem 9 yields an IR-calc lower bound.

▶ **Theorem 14.** *Any linear product requires superpolynomial-size* IR-calc *refutations.*

4.3 Separations and propositional hardness

As a further application of Theorem 9, we prove an interesting theorem with clear relevance to QBF solving.

First, consider a PCNF $F := \mathcal{Q} \cdot \phi$ that has a countermodel S. The elements of the range of S are all total assignments to the universal variables of F, and it should be clear that instantiating each clause in ϕ by each element of $\mathrm{rng}(S)$ gives rise to an unsatisfiable set of clauses in annotated variables. Let us denote this set $\psi := \mathrm{inst}(\phi, \mathrm{rng}(S))$, and say that F *expands* to ψ. Further, let us say that a PCNF family $\mathcal{F}$ *expands to a CNF family* f if and only if $\mathcal{F}(n)$ expands to $f(n)$, for each natural number n.

An immediate corollary to Theorem 9 is that any bounded PCNF family with polynomial-size IR-calc refutations must have polynomial strategy size; hence any such family expands to a CNF family of polynomial-size. This observation leads to the following theorem.

▶ **Theorem 15.** *Let $\mathcal{F}$ be a bounded PCNF family separating* IR-calc *from* $\forall$Exp+Res*. Then $\mathcal{F}$ expands to a polynomial-size CNF family requiring superpolynomial-size resolution refutations.*

Proof. Let $\mathcal{F}(n) := \mathcal{Q}_\mathcal{F}(n) \cdot \phi_\mathcal{F}(n)$. Since $\mathcal{F}$ has polynomial-size IR-calc refutations, $\nabla_\mathcal{F}$ is polynomially bounded, by Theorem 9. Hence, there exist countermodels $S(n)$ for $\mathcal{F}(n)$ for which $|\mathrm{rng}(S(n))|$ is polynomially bounded, and the number of literals in the CNF $f(n) := \mathrm{inst}(\phi_\mathcal{F}(n), \mathrm{rng}(S(n)))$ is polynomially bounded. Therefore, the function $f : n \mapsto f(n)$ is a CNF family. Observe that every clause in $f(n)$ may be downloaded as an axiom in a $\forall$Exp+Res derivation from $\mathcal{F}(n)$, and that $\mathcal{F}$ requires superpolynomial-size $\forall$Exp+Res refutations. It follows that polynomial-size resolution refutations of f do not exist. ◀

The import of Theorem 15. The essence of the result can perhaps be captured as follows: if the lower bound is not derived from propositional hardness, a separation of IR-calc from ∀Exp+Res must be due to an unbounded family of PCNFs. In the spirit of [13], it is natural to label this kind of separation as *genuine*, since the ∀Exp+Res lower bound is due to a large expansion, rather than a large number of resolution steps.

Moreover, since IR-calc simulates the well-studied QBF proof system Q-resolution (Q-Res [29]), Theorem 15 holds when IR-calc is replaced by Q-Res. Thus, a 'genuine separation' of Q-Res from ∀Exp+Res requires an unbounded PCNF family.

As the theoretical models of ∀Exp+Res and Q-Res underpin the two major paradigms in QBF practice – expansion-based solving [27] and QCDCL [24] – Theorem 15 has a clear practical import. A typical QBF expansion solver will use a SAT solver as an oracle, assuming that SAT calls are inexpensive. According to Theorem 15, if bounded formulas separating Q-Res from ∀Exp+Res exist, they may still be easy for an expansion-based algorithm given access to a SAT oracle, and hence offer no insight into how to improve the algorithm.

5 The Weight Theorem: conquering unbounded families

In this section, we extend the lower-bound technique to cover unbounded PCNF families. Since the technical details are quite demanding, the proof of the main theorem is preceded by a brief overview of the technique. We conclude with an application: a very short proof of hardness for what is arguably the most famous PCNF family.

5.1 Outline of the technique

We invite the reader to consider once again the example PCNF family $\mathcal{J}$ (Definition 4) from the previous section. That family has exponential strategy size and linear-size IR-calc refutations. This illustrates that the responses from the extracted strategy do not always appear as annotations in an IR-calc refutation. However, with careful analysis, we can show that *certain portions of the responses always will*.

Our method makes use of a particular class of assignments: assignments to all existentials except those in the final block. We call such assignments *restrictors*.

▶ **Definition 16.** Let Z be the rightmost block of a PCNF F. Any total assignment to the variables $\mathrm{vars}_\exists(F) \setminus Z$ is a *restrictor* of F.

Now, take a refutation π of a PCNF F and select a restrictor α. First, apply strategy extraction to π, and consider the response $S(\alpha)$ in the extracted strategy S. Then compare this response with the final annotation τ_α of the restricted refutation $\pi[\alpha]$. On the one hand, the definition of strategy extraction ensures that the literals in τ_α are a subset of the response $S(\alpha)$. We combine this with a proof that certain *critical variables* must occur in τ_α. As a result, we obtain a subset of the response to α, called the *critical response*, that must be contained in the annotation τ_α. This is the central observation of our method, depicted in Figure 2. Note that τ_α occurs also as an annotation in the original refutation.

Proof of the Weight Theorem. The *critical variables* of a PCNF are those universals that appear in every subset of the matrix that is false under the quantifier prefix. The projection of a restrictor's response to its critical variables is termed the *critical response*.

▶ **Definition 17.** Let S be a countermodel for a false PCNF $F := \mathcal{Q} \cdot \phi$, and let α be a restrictor of F. The *critical variables* of F are the universal variables appearing in every CNF ϕ' for which (a) $\phi' \subseteq \phi$ and (b) $\mathcal{Q} \cdot \phi'$ is false. The *critical response* to α with respect to S and F is the projection of $S(\alpha)$ to the critical variables of $F[\alpha]$.

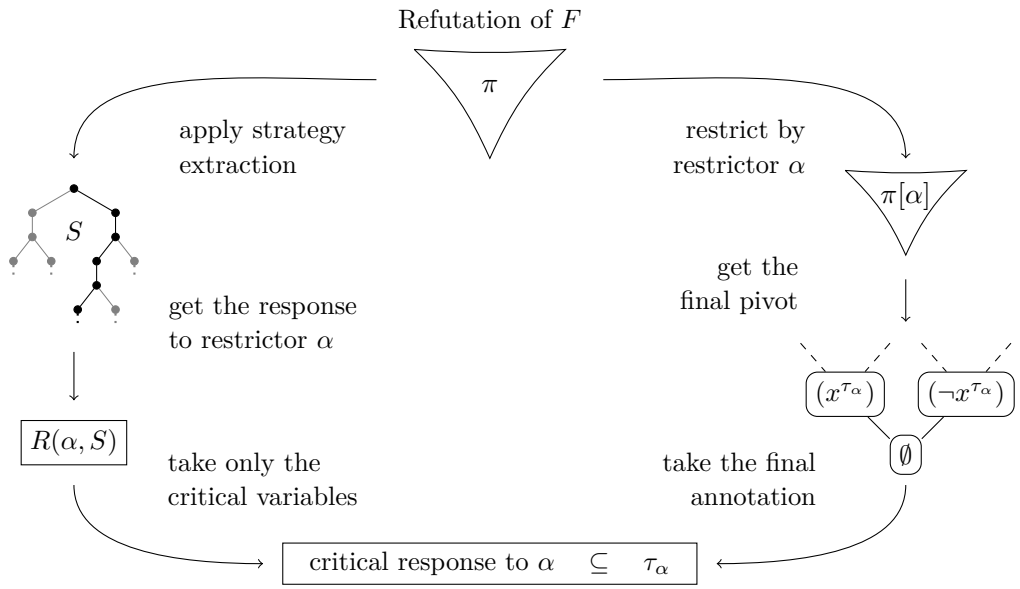


Figure 2 Depiction of the central observation of our lower-bound technique. The final statement is proved in Lemma 18.

The key notion in our argument is the following relationship between the critical response to a restrictor and the final annotation of the restricted refutation.

▶ **Lemma 18.** *Let S be the extracted strategy for a* IR-calc *refutation π of a PCNF F. Then, for each restrictor α of F, the final annotation of $\pi[\alpha]$ contains the critical response to α with respect to S and F.*

Proof sketch. The lemma is vacuously true if F contains no universal variables, so we assume otherwise. Let α be a restrictor of F, and let $\tau[\alpha]$ be the final annotation of $\pi[\alpha]$. In combination with Proposition 1, the fact that $F[\alpha]$ has a Π_2 prefix is enough to deduce that $\mathrm{vars}(\tau_\alpha)$ contains the critical variables of $F[\alpha]$. Hence, the lemma follows from the claim that $\tau_\alpha \subseteq S(\alpha)$, a fairly straightforward consequence of the definition of strategy extraction, and Propositions 1 and 2. ◀

Since the final annotation of $\pi[\alpha]$ appears also in π, any k mutually inconsistent critical responses give rise to k distinct annotations in π. For that reason, given a winning $\forall$-strategy S, we define the *critical response graph* that has a vertex for each critical response and an edge between each inconsistent pair. Hence, as we prove subsequently, the number of distinct annotations in a refutation is lower bounded by the clique number of the critical response graph for the extracted strategy. The clique number of a graph G is denoted $\omega(G)$.

▶ **Definition 19.** Let S be a countermodel for a PCNF F. The *critical response graph* of S with respect to F is the undirected graph $G(S, F)$ defined as follows: (a) For each restrictor α of F, $G(S, F)$ has a vertex labelled with the critical response to α with respect to S and F; (b) $G(S, F)$ has an edge between two vertices if and only if their labels are inconsistent.

▶ **Lemma 20.** *Let S be the strategy extracted from an* IR-calc *refutation π of a PCNF F. Then there are at least $\omega(G(S, F))$ distinct annotations in π.*

Proof. Let $k := \omega(G(S, F))$, and let $\alpha_1, \ldots, \alpha_k$ be restrictors of F whose critical responses (with respect to S and F) are pairwise inconsistent. For each $i \in [k]$, the final annotation τ_{α_i}

of $\pi[\alpha_i]$ contains the critical response to α_i, by Lemma 18, and τ_{α_i} appears as an annotation in π (an existential restriction of π preserves any annotation that is not deleted). Hence, for each $i, j \in [k]$ with $i \neq j$, τ_{α_i} and τ_{α_j} are distinct annotations appearing in π. ◀

An IR-calc proof is at least as large as the number of distinct annotations; hence, the minimal clique number of a critical response graph for a countermodel yields a refutation-size lower bound. This motivates the following definition, in which we define the *weight* of a PCNF F, denoted $\mu(F)$, to be equal to this minimal clique number.

▶ **Definition 21.** The *weight* $\mu(F)$ of a false PCNF F is the minimum value of $\omega(G(S, F))$ over the countermodels S of F.

The main result of this section, the Weight Theorem, is almost immediate from Lemma 20.

▶ **Theorem 22** (Weight Theorem). *The size of any* IR-calc *refutation of a PCNF F is at least the weight of F.*

Proof. Let S be the strategy extracted from a refutation π of F. Since S is a winning $\forall$-strategy by Proposition 3, the weight of F is at most $\omega(G(S, F))$. By Lemma 20, at least $\omega(G(S, F))$ distinct annotations, and at least as many distinct literals, appear in π. ◀

5.2 Application to the formulas of Kleine Büning et al.

The final application of our framework is to the familiar QBFs introduced in [29] which occupy a central place in the QBF proof complexity literature (e.g. [21, 8, 2, 34]; the original formulas from [29] are called Φ_t and appear there in the proof of Theorem 3.2). We state the formulas and then prove that they have exponential weight. The IR-calc lower bound follows immediately, by the Weight Theorem (Theorem 22).

▶ **Definition 23** ([29]). Let $\mathcal{K}$ be the PCNF family defined by $\mathcal{K}(n) := \mathcal{Q}_{\mathcal{K}}(n) \cdot \phi_{\mathcal{K}}(n)$, where

$$\begin{aligned} \mathcal{Q}_{\mathcal{K}}(n) &:= \exists x_1 y_1 \forall u_1 \cdots \exists x_n y_n \forall u_n \exists t_1 \cdots t_n \,, \\ \phi_{\mathcal{K}}(n) &:= \{(\neg x_1, \neg y_1), (x_n, u_n, \neg t_1, \ldots, \neg t_n), (y_n, \neg u_n, \neg t_1, \ldots, \neg t_n)\} \\ &\quad \textstyle\bigcup_{i=1}^{n-1} \{(x_i, u_i, \neg x_{i+1}, \neg y_{i+1}), (y_i, \neg u_i, \neg x_{i+1}, \neg y_{i+1})\} \\ &\quad \textstyle\bigcup_{i=1}^{n} \{(u_i, t_i), (\neg u_i, t_i)\} \,. \end{aligned}$$

▶ **Lemma 24.** *For each $n \in \mathbb{N}$, the weight of $\mathcal{K}(n)$ is at least 2^n.*

Proof sketch. Consider the set A of restrictors of $\mathcal{K}(n)$ that contain exactly one of $\neg x_i$ and $\neg y_i$ for each $i \in [n]$, and let $\alpha \in A$. For any countermodel S of $\mathcal{K}(n)$, the gameplay implies that $\neg u_i \in S(\alpha) \Leftrightarrow \neg x_i \in \alpha$ and $u_i \in S(\alpha) \Leftrightarrow \neg y_i \in \alpha$, for each $i \in [n]$. Moreover, it can be verified that $\mathrm{vars}_\forall(\mathcal{K}(n))$ are all critical in $\mathcal{K}(n)[\alpha]$. It follows that every total assignment to the universals is the critical response to some restrictor in A. Hence, the critical response graph $G(S, \mathcal{K}(n))$ has a 2^n-clique. ◀

Applying the Weight Theorem concludes a very short proof of this historic QBF result.

▶ **Theorem 25** ([29, 8]). *The family $\mathcal{K}(n)$ requires exponential-size* IR-calc *refutations.*

6 Conclusions

We introduced the first technique for genuine QBF lower bounds in expansion systems. As applications, we proved exponential IR-calc lower bounds for a new class of formula families, and produced greatly simplified proofs of two known hardness results. Whereas our work on unbounded families was based on restrictions up to the penultimate existential block, the technique could be explored in greater generality by considering restrictions up to the i^{th} block. We also applied the technique to prove that any bounded separation of IR-calc from $\forall$Exp+Res is due to a non-genuine lower bound. It remains an open problem whether such a bounded separation exists.

References

1 Valeriy Balabanov, Jie-Hong Roland Jiang, Mikoláš Janota, and Magdalena Widl. Efficient extraction of QBF (counter)models from long-distance resolution proofs. In *Conference on Artificial Intelligence (AAAI)*, pages 3694–3701, 2015.

2 Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 154–169, 2014.

3 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow - resolution made simple. *Journal of the ACM*, 48(2):149–169, 2001.

4 Marco Benedetti and Hratch Mangassarian. QBF-based formal verification: Experience and perspectives. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 5(1-4):133–191, 2008.

5 Olaf Beyersdorff, Joshua Blinkhorn, and Luke Hinde. Size, cost, and capacity: A semantic technique for hard random QBFs. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:35, 2017.

6 Olaf Beyersdorff, Ilario Bonacina, and Leroy Chew. Lower bounds: From circuits to QBF proof systems. In *ACM Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 249–260, 2016.

7 Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. On unification of QBF resolution-based calculi. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 81–93, 2014.

8 Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. Proof complexity of resolution-based QBF calculi. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 30, pages 76–89, 2015.

9 Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla. Feasible interpolation for QBF resolution calculi. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 180–192, 2015.

10 Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla. Are short proofs narrow? QBF resolution is not simple. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 15:1–15:14, 2016.

11 Olaf Beyersdorff, Leroy Chew, and Karteek Sreenivasaiah. A game characterisation of tree-like Q-resolution size. *Journal of Computer and System Sciences (in press)*, 2017.

12 Olaf Beyersdorff, Nicola Galesi, and Massimo Lauria. A characterization of tree-like resolution size. *Information Processing Letters*, 113(18):666–671, 2013.

13 Olaf Beyersdorff, Luke Hinde, and Ján Pich. Reasons for hardness in QBF proof systems. In *Conference on Foundations of Software Technology and Theoretical Computer Science (FCTTCS)*, 2017. Preprint available at ECCC, TR17-044.

14 Samuel R. Buss. Towards NP-P via proof complexity and search. *Ann. Pure Appl. Logic*, 163(7):906–917, 2012.

15 Michael Cashmore, Maria Fox, and Enrico Giunchiglia. Partially grounded planning as quantified Boolean formula. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2013.

16 Hubie Chen. Proof complexity modulo the polynomial hierarchy: Understanding alternation as a source of hardness. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 94:1–94:14, 2016.

17 Stephen A. Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Cambridge University Press, Cambridge, 2010.

18 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979.

19 William Craig. Linear reasoning. A new form of the Herbrand-Gentzen Theorem. *J. Symb. Log.*, 22(3):250–268, 1957.

20 Nachum Dershowitz, Ziyad Hanna, and Jacob Katz. Space-efficient bounded model checking. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 502–518. Springer, 2005.

21 Uwe Egly. On stronger calculi for QBFs. In *Theory and Applications of Satisfiability Testing (SAT)*, pages 419–434, 2016.

22 Uwe Egly, Martin Kronegger, Florian Lonsing, and Andreas Pfandler. Conformant planning as a case study of incremental QBF solving. *Annals of Mathematics and Artificial Intelligence*, 80(1):21–45, 2017.

23 Uwe Egly, Florian Lonsing, and Magdalena Widl. Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In *International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, pages 291–308, 2013.

24 Enrico Giunchiglia, Paolo Marin, and Massimo Narizzano. Reasoning with quantified Boolean formulas. In *Handbook of Satisfiability*, pages 761–780. IOS Press, 2009.

25 Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. QBF reasoning on real-world instances. In *International Conference on Theory and Applications of Satisfiability Testing (SAT), online proceedings*, 2004.

26 Mikolás Janota, William Klieber, Joao Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. *Journal of Artificial Intelligence*, 234:1–25, 2016.

27 Mikoláš Janota and João Marques-Silva. Expansion-based QBF solving versus Q-resolution. *Theoretical Computer Science*, 577:25–42, 2015.

28 Charles Jordan and Lukasz Kaiser. Experiments with reduction finding. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 192–207, 2013.

29 Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified Boolean formulas. *Information and Computation*, 117(1):12–18, 1995.

30 Roman Kontchakov, Luca Pulina, Ulrike Sattler, Thomas Schneider, Petra Selmer, Frank Wolter, and Michael Zakharyaschev. Minimal module extraction from DL-lite ontologies using QBF solvers. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 836–841. AAAI Press, 2009.

31 Jan Krajícek. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *Journal of Symbolic Logic*, 62(2):457–486, 1997.

32 Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*, volume 60 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, Cambridge, 1995.

33 Andrew C. Ling, Deshanand P. Singh, and Stephen Dean Brown. FPGA logic synthesis using quantified boolean satisfiability. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 444–450, 2005.

34 Florian Lonsing, Uwe Egly, and Martina Seidl. Q-resolution with generalized axioms. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 435–452, 2016.

35 Hratch Mangassarian, Andreas G. Veneris, and Marco Benedetti. Robust QBF encodings for sequential circuits with applications to verification, debug, and test. *IEEE Transactions on Computers*, 59(7):981–994, 2010.

36 Hratch Mangassarian, Andreas G. Veneris, Sean Safarpour, Marco Benedetti, and Duncan Exon Smith. A performance-driven QBF-based iterative logic array representation with applications to verification, debug and test. In *International Conference on Computer-Aided Design (ICCAD)*, pages 240–245, 2007.

37 Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Long distance Q-resolution with dependency schemes. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 500–518, 2016.

38 Pavel Pudlák and Russell Impagliazzo. A lower bound for DLL algorithms for sat (preliminary version). In *Symposium on Discrete Algorithms*, pages 128–136, 2000.

39 Jussi Rintanen. Asymptotically optimal encodings of conformant planning in QBF. In *National Conference on Artificial Intelligence (AAAI)*, pages 1045–1050. AAAI Press, 2007.

40 Nathan Segerlind. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 13(4):417–481, 2007.

41 Stefan Staber and Roderick Bloem. Fault localization and correction with QBF. In *International Conference on Theory and Applications of Satisfiability Testing (SAT) 2007*, pages 355–368, 2007.

42 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *Annual Symposium on Theory of Computing*, pages 1–9. ACM, 1973.

Efficient Oracles and Routing Schemes for Replacement Paths

Davide Bilò
Università di Sassari, Italy
davidebilo@uniss.it

Keerti Choudhary
Department of CSE, I.I.T. Kanpur, India
keerti@cse.iitk.ac.in

Luciano Gualà
Università di Roma "Tor Vergata", Italy
guala@mat.uniroma2.it

Stefano Leucci
ETH Zürich, Switzerland
stefano.leucci@inf.ethz.ch

Merav Parter
CSAIL, MIT
meravparter@gmail.com

Guido Proietti
DISIM, Università dell'Aquila, Italy and
Istituto di Analisi dei Sistemi ed Informatica, CNR, Roma, Italy
guido.proietti@univaq.it

Abstract

Real life graphs and networks are prone to failure of nodes (vertices) and links (edges). In particular, for a pair of nodes s and t and a failing edge e in an n-vertex unweighted graph $G = (V(G), E(G))$, the *replacement path* $\pi_{G-e}(s,t)$ is a shortest $s-t$ path that avoids e. In this paper we present several efficient constructions that, for every $(s,t) \in S \times T$, where $S, T \subseteq V(G)$, and every $e \in E(G)$, maintain the collection of all $\pi_{G-e}(s,t)$, either *implicitly* (i.e., through compact data structures a.k.a. *distance sensitivity oracles* (DSO)), or *explicitly* (i.e., through sparse subgraphs a.k.a. *fault-tolerant preservers* (FTP)). More precisely, we provide the following results:

(1) *DSO:* For every $S, T \subseteq V(G)$, we construct a DSO for maintaining $S \times T$ distances under single edge (or vertex) faults. This DSO has size $\tilde{O}(n\sqrt{|S||T|})$ and query time of $O(\sqrt{|S||T|})$. At the expense of having *quasi-polynomial* query time, the size of the oracle can be improved to $\tilde{O}(n|S| + |T|\sqrt{|S|n})$, which is optimal for $|T| = \Omega(\sqrt{n|S|})$. When $|T| = \Omega(n^{\frac{3}{4}}|S|^{\frac{1}{4}})$, the construction can be further refined in order to get a polynomial query time. We also consider the *approximate additive* setting, and show a family of DSOs that exhibits a tradeoff between the additive stretch and the size of the oracle. Finally, for the meaningful single-source case, the above result is complemented by a lower bound conditioned on the Set-Intersection conjecture. This lower bound establishes a *separation* between the oracle and the subgraph settings.

(2) *FTP:* We show the construction of a *path-reporting* DSO of size $\tilde{O}(n^{4/3}(|S||T|)^{1/3})$ reporting $\pi_{G-e}(s,t)$ in $O(|\pi_{G-e}(s,t)| + (n|S||T|)^{1/3})$ time. Such a DSO can be transformed into a FTP having the same size, and moreover it can be elaborated in order to make it optimal (up to a poly-logarithmic factor) both in space and query time for the special case in which $T = V(G)$. Our FTP improves over previous constructions when $|T| = O(\sqrt{|S|n})$ (up to inverse poly-logarithmic factors).

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 13; pp. 13:1–13:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

(3) *Routing and Labeling Schemes:* For the well-studied single-source setting, we present a novel *routing scheme*, that allows to route messages on $\pi_{G-e}(s,t)$ by using edge labels and routing tables of size $\tilde{O}(\sqrt{n})$, and a header message of poly-logarithmic size. We also present a labeling scheme for the setting which is optimal in space up to constant factors.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases Fault Tolerant, Shortest Path, Oracle, Routing

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.13

1 Introduction

1.1 Motivation

Shortest path in graphs is perhaps one of the most classical concepts in network algorithms. As real life networks are prone to failures, much attention has been devoted, recently, for studying *replacement paths*, namely, shortest paths that avoid failed edges or vertices.

A traditional objective in shortest path research is to reduce the size of the distance representation. One common way to do so is to use sparse graph *spanners*, that is a spanning subgraph of the original graph using possibly few edges while preserving some distance information. In the context of fault tolerance, Peleg and Parter [24] introduced the notion of *FT-BFS trees*, namely sparse subgraphs that contain a collection of all replacement paths from a given source s that avoids a single edge or vertex in the graph. For an n-vertex unweighted graph $G = (V(G), E(G))$, [24] showed a simple construction of FT-BFS subgraphs with $O(n^{3/2})$ edges. For the case of multiple sources $S \subseteq V$, they showed the construction of an FT-BFS for each $s \in S$ with $O(n^{3/2}\sqrt{|S|})$ edges. Albeit being optimal in space, FT-BFS structures $H \subseteq G$ are lacking some useful properties such as fast reporting of $s-t$ distances in $G - e = (V(G), E(G) \setminus \{e\})$ or being able to route messages along the replacement paths. For instance, to return the distance between the source vertex s and any other vertex t of the graph, following a failure of e, the best one can do with FT-BFS structure is to run a Dijkstra's algorithm in $H - e$ rather than $G - e$.

Our goal in this paper is to devise more structured representations of replacement paths that have useful applications in communication networks.[1] We present efficient constructions of data structures that enjoy not only optimal space (like FT-BFS subgraphs) but also have additional desired attributes, e.g., allowing fast extraction of distances; balanced information spreading in the network; and routing on replacement paths using small routing tables.

In principle, storing the replacement paths in data structures might be more space efficient than using a subgraph of the original network. Unfortunately, here this is not the case; by using standard tools [11, 22, 1], one can show that the lower bound of $\Omega(n^{3/2}\sqrt{|S|})$ edges for FT-BFS structures for $S \times V$ distances applies against *any* kind of replacement paths representation, and not just subgraphs. Our starting point is:

> *There are bad n-vertex graph families, for which any representation allowing for the return of all the $S \times V$ post-failure distances must have size $\Omega(n^{3/2}\sqrt{|S|})$* bits.

[1] We focus on single edge failures and undirected graphs, although most of the results extend to single vertex failures and directed graphs as well.

Unlike the sourcewise scenario that has been studied thoroughly in the subgraph setting, almost nothing is known for the more general $S \times T$ case, i.e., where T is not necessarily V, but for the fault-free framework [20, 15]. We fill some of that gap here and provide tools that go beyond the sourcewise setting.

1.2 Contribution

We provide a comprehensive study of several space aspects for replacement paths. We consider three fundamental data structures for maintaining shortest paths: distance sensitivity oracles, labeling schemes and compact routing schemes. Roughly speaking, *distance sensitivity oracle* is a compact data structure that can also report distances fast; *labeling scheme* is a more structured type of distance sensitivity oracles in which (hopefully) the same amount of distance information is now spread evenly in the network and hence the memory load per vertex is bounded; Finally a *compact routing scheme* is a distributed algorithm that sends messages from s to t along some short path. The next hop is computed by using the information at the message headers as well as the routing table stored at the current vertex.

Distance Sensitivity Oracles (DSO) and Labeling Schemes

For an n-vertex unweighted graph $G = (V(G), E(G))$, subsets $S, T \subseteq V$, an $S \times T$ DSO is a compact data structure that answers efficiently queries of the form (s, t, e)*: Return the distance between* $s \in S$ *and* $t \in T$ *when the edge* e *fails.* Our main results are the following:

- A polynomial time constructable $S \times T$ DSO of size $\tilde{O}(n\sqrt{|S||T|})$ and query time $O(\sqrt{|S||T|})$. If *quasi-polynomial* query time is allowed, then the size of such oracle can be improved to $\tilde{O}(n|S| + |T|\sqrt{|S|n})$, which we will show to be optimal for $|T| = \Omega(\sqrt{n|S|})$. Moreover, when $|T| = \Omega(n^{\frac{3}{4}}|S|^{\frac{1}{4}})$, the construction can be further refined in order to get a polynomial query time.
- A polynomial time constructable family of *approximate* $S \times T$ DSOs, returning in constant time a distance stretched by an *additive* term which decreases as soon as the size of the oracle increases. In particular, for $|S| = O(\sqrt{n})$, we can obtain an oracle of size $\tilde{O}(n^{3/2})$ and additive distortion $\tilde{O}(\sqrt{n})$.
- A *path-reporting* DSO of size $\tilde{O}(n^{4/3}(|S||T|)^{1/3})$ returning $\pi_{G-e}(s,t)$ in $O(|\pi_{G-e}(s,t)| + (n|S||T|)^{1/3})$ time. Such a DSO can be transformed into an $S \times T$ *fault-tolerant preserver* (FTP) (i.e., a subgraph of G maintaining all the $S \times T$ shortest paths after any edge failure) having the same size. Thus, our FTP improves over the multi-source preserver provided in [24] as soon as $|T| = O(\sqrt{|S|n})$ (up to inverse poly-logarithmic factors). Finally, for the remarkable case in which $T = V(G)$, it can be elaborated in order to get a DSO having size $\tilde{O}(n\sqrt{n|S|})$ and reporting $\pi_{G-e}(s,t)$ in $O(|\pi_{G-e}(s,t)|)$ time; this construction represents the oracle counterpart of the multi-source preserver provided in [24], and thus it has not only optimal size (up to a poly-logarithmic factor), but it also allows to retrieve a shortest path in optimal time.
- Let $\epsilon \in (0, 1]$ be any fixed constant; we show that conditioned on the Set-Intersection Conjecture [26], any $\{s\} \times V$ DSO with constant query time and additive distortion $d = O(n^{1-\epsilon})$ must use $\tilde{\Omega}\left(n^{\frac{3}{2}\epsilon}\right)$ bits of memory.

Concerning the first result, our construction in fact gives a tradeoff between the query time and the size of the oracle. Note that prior to our construction, for the single-source setting, a trivial query time was $O(n^{3/2})$ by running Dijkstra on the FT-BFS structure with $O(n^{3/2})$ edges. For the $S \times T$ setting, the trivial query time was $O(\sqrt{|S|}n^{3/2})$, using the FT-BFS construction of [24] for multiple sources S.

Concerning the lower bound for the single-source setting, it compares favorably with the non-conditional lower-bounds given in [25]. Indeed, it improves the range of additive distortions for which no linear-size $\{s\} \times V$ DSO can exist from $d = O(\log n)$ to $d = O(n^{\frac{1}{3}-\epsilon})$, for any constant $\epsilon > 0$. Moreover, it shows that for any $d = O(1)$, $\tilde{\Omega}(n^{\frac{3}{2}})$ bits are needed by any $\{s\} \times V$ DSO with constant query time. This is in contrast with the 4-additive FT-BFS structure of size $O(n^{\frac{4}{3}})$ given in [25] thus establishing that designing a corresponding oracle is harder than its FT-BFS counterpart. Notice also that for exact distances (i.e., $d = 0$) this lower bound still allows for the existence of a (single-source) DSO having size $O(n^{\frac{3}{2}})$ and constant query time. We regard the problem of finding the best query time for an optimal-size DSO as an interesting remaining open problem. Due to space limitations the discussion of our lower bound, as well as the proof of several statements, will be provided in the full version of the paper.

Single-Source Labeling Schemes. Labeling schemes are special type of a "balanced" distance oracle with the benefit of having the distance information evenly distributed between all the nodes in the network. Here we obtain a space-optimal (up to constant factors) labeling scheme for the meaningful *single-source to all-destinations* case. It consists of a label with $\tilde{O}(\sqrt{n})$ bits for each node, which allows to compute $|\pi_{G-e}(s,t)|$, by simply looking at the label of t and of the end-vertices of the failing edge e.

Single-Source Routing Scheme

A routing scheme for a given source s is a distributed mechanism that, for the failure of any edge $e \in E$, can deliver packets of information from s to any other node t of the network along the corresponding replacement path. This is done by storing compact *routing tables* at each node, by assigning labels to edges, and finally by adding a short header to the message containing information about the target t and the failing edge e. Our key observation is that every replacement path can be decomposed into two (fault-free) *tree* paths connected by an edge, as shown in [21]. By combining the routing schemes for trees of Thorup and Zwick [29] along with our labeling scheme, we can provide the following:

- A scheme for routing packets from a source s along shortest paths with poly-logarithmic headers and $\tilde{O}(\sqrt{n})$-size routing tables and edge labels.

1.3 Additional Related Work

In this work, we mainly consider exact distances under faults. In the literature, many related settings have been studied thoroughly as discussed next.

Single source approximate shortest paths avoiding any failed vertex. Baswana and Khanna [3] showed that for the undirected unweighted graph $G = (V, E)$, one can construct a subgraph H with $O(n \log n/\epsilon^3)$ edges satisfying that $\texttt{dist}(s, t, H-e) \leq (1+\epsilon)\,\texttt{dist}(s, t, G-e)$ for every $t \in V(G), e \in E(G)$. They also provide a DSO of the same size that can report these distances or even the paths in optimal time. This was later extended to the weighted case, for both the subgraph [7] and the oracle setting [8]. *Multiple faults* have been studied in [9, 25] and structures with *additive* stretch have been studied in [23, 6].

Distance sensitivity oracles (for all pairs). In a seminal work, Demetrescu et al. [17] showed that given a directed weighted graph G of size n, it is possible to construct in time $\tilde{O}(mn^2)$ a DSO of size $O(n^2 \log n)$ capable of answering distance queries in $O(1)$ time in

the presence of a single failed edge or vertex. The preprocessing time was then improved to $\tilde{O}(mn)$, with unchanged size and query time [5]. Grandoni and Williams [19] presented the first DSO that achieves simultaneously subcubic preprocessing time and sublinear query time for directed graphs with bounded integer edge weights. A dual failure fault tolerant DSO of size $O(n^2 \log^3 n)$ and $O(\log n)$ query time was presented in [18]. The f faults case was studied in [30, 13].

FT distance labels and compact routing schemes. Label-based fault-tolerant routing schemes for graphs of bounded clique-width are presented in [16]. To route from s to t, the source needs to specify the labels $\lambda(s)$ and $\lambda(t)$ and the set of failures F, and the scheme efficiently calculates the shortest path between s and t that avoids F. For an n-vertex graph of tree-width or clique-width k, the constructed labels are of size $O(k^2 \log^2 n)$. Turning to general graphs, FT compact routing schemes were first considered in [14], for up to two edge failures. Further work considered multiple failures [12] and $(1+\epsilon)$ approximation [2].

Set intersection and distance oracles. The set intersection problem has several related variants and has been widely used to provide conditional lower bounds on the space and query time of distance oracles. The folklore conjecture for set intersection states that, given n sets of cardinality polylogarithmic in n, answering a set intersection query in constant time requires $\Omega(n^2)$ space. For the connection between distance oracles and various variants of the set intersection problem, see [28, 26, 27]. In this paper we provide the first connection between distance *sensitivity* oracles and the set intersection problem.

2 Preliminaries and Notations

Let $G = (V(G), E(G))$ be a directed or undirected graph on n vertices with $S \subseteq V(G)$ as the source set and $T \subseteq V(G)$ as the destination set. Let H be a subgraph of G. We use H^R to denote the graph obtained by reversing all edge directions of H (if H is undirected then H^R is same as H). For any vertex w, let $\mathcal{T}_{w,H}$ be the shortest path tree of H rooted at w, and $\mathcal{T}^R_{w,H}$ be the shortest path tree of H^R rooted at w. When H is same as G, we can as well use the notions $\mathcal{T}_w$ and $\mathcal{T}^R_w$. We will denote by $\pi_H(u,v)$ the shortest path between the two vertices u and v in H, and by $d_H(u,v)$ its length, i.e., the distance between u and v in H. Moreover, whenever $H = G$, we will omit the subscript. Given a set $F \subseteq E(G)$ of edges, we will denote by $G - F$ the subgraph of G obtained by removing the edges in F from $E(G)$. For the sake of simplicity we might slightly abuse the notation and write $G - e$ instead of $G - \{e\}$ when $F = \{e\}$. Given a simple path P, we denote by $|P|$ its *size*, i.e., the number of its edges. Moreover, if P traverses the vertices u and v in this order, we denote by $P(u,v)$ the subpath of P between u and v (endpoints included). For any non-negative integer i, we define $P[-i]$ to be the path containing the last $\min\{|P|, i\}$ edges of P. Given any two paths P and Q with last vertex of P same as the first vertex of Q, we use $P{::}Q$ to denote the path formed by concatenating paths P and Q.

Given a tree $\mathcal{T}$ and any two vertices $a, b \in \mathcal{T}$, we use the notation $\mathcal{T}(a,b)$ to denote the path from a to b in tree $\mathcal{T}$. Throughout the paper we use $\tilde{O}(f(x))$ (resp. $\tilde{\Omega}(f(x))$) as a shorthand for $O(f(x)\,\mathrm{polylog} f(x))$ (resp. $\Omega(f(x)/\mathrm{polylog} f(x))$). Below we state a lemma that will be crucially used in our fault tolerant data structures.

▶ **Lemma 1.** *Let G be an undirected unweighted graph, and let $L \in [5, n/\log n]$ and $\mathcal{P} = \{\pi(u,v) \mid u, v \in V(G), d(u,v) \geq L \log n\}$ be the family of shortest paths in G having length at least $L \log n$. Then (i) In expected polynomial time we can compute a subset R of $V(G)$*

with $O(n/L)$ vertices such that $R \cap V(P) \neq \emptyset$ for each path $P \in \mathcal{P}$; (ii) We can also have a deterministic polynomial time construction for set R that intersects each path in $\mathcal{P}$, such an R contains $O\left(\frac{n}{L} \log n\right)$ vertices.

Although Lemma 1 allows for both randomized and deterministic constructions, in the rest of the paper, we will only focus on the randomized case, as however this will only differ up to logarithmic factors in the query time and the size of our solutions.

We assume edge weights are slightly perturbed by adding a small noise so that edge-weights are always positive and between any two vertices x, y there is exactly one shortest path. This will help us to uniquely define $\pi_H(x, y)$ for any subgraph H of G. When we focus on undirected graphs, we assume perturbation is small enough so that for any simple path P between x and y of weighted length λ, we have $|P| = \lfloor \lambda \rfloor$.

3 Distance Sensitivity Oracle

The basic building block in our construction is an $W \times W$ DSO that reports, in $O(1)$ time, the distance between any pair of vertices in $W \subseteq V(G)$. This will be used to obtain our $S \times T$ oracle.

3.1 Distance Sensitivity Oracle for $W \times W$

As an input we are given a set W of vertices in a directed or undirected *weighted* graph G. We will use ideas similar to the ones of the edge/vertex fault tolerant $V \times V$ oracle of [17]. For the sake of simplicity we only discuss the edge-failure case, but our results naturally extend to the vertex failures as well. Our data structure stores the following information:

1. For each $w \in W$, it stores:
   - An incoming and an outgoing shortest path tree rooted at w, i.e. trees $\mathcal{T}_w$ and $\mathcal{T}_w^R$;
   - The pre-order and post-order numbering, and depth of each $v \in V$ in $\mathcal{T}_w$ and $\mathcal{T}_w^R$;
   - A level ancestor data structure for trees $\mathcal{T}_w$ and $\mathcal{T}_w^R$, namely a data structure able to return in $O(1)$ time the k-th ancestor of a node, for any $k \geq 1$ [4];
   - The distances $d(w, v)$ and $d(v, w)$, where $v \in V$.
2. For every vertex pair $(s, t) \in (W \times V) \cup (V \times W)$ and every integer $0 \leq i \leq \log |\pi(s, t)|$:
   - B1(s, t, i) stores the distance $d_{G-e}(s, t)$, where $e = (u, v)$ is an edge lying on $\pi(s, t)$ and satisfying $|\pi(s, u)| = 2^i$;
   - B2(s, t, i) stores the distance $d_{G-\pi(u,v)}(s, t)$, where u, v are vertices on $\pi(s, t)$ and satisfying (i) $|\pi(u, v)| = 2^i$, and (ii) $|\pi(s, u)| = 2^i$.

We now explain the query process. Let $(s, t) \in W \times W$ be a query pair and $e = (u, v)$ be a failing edge lying on $\pi(s, t)$. (Whether e lies on $\pi(s, t)$ or not can be verified in constant time using pre-order and post-order numbering, and depth of vertices in $\mathcal{T}_w$ and $\mathcal{T}_w^R$, $w \in W$). Let i_0 and j_0 be greatest integers satisfying $2^{i_0} \leq |\pi(s, u)|$ and $2^{j_0} \leq |\pi(u, t)|$. Let s', t' be vertices on $\pi(s, t)$ such that $|\pi(s', u)| = 2^{i_0}$ and $|\pi(v, t')| = 2^{j_0}$. (See Figure 1). These vertices can be computed in constant time by using the level ancestor data structure on shortest path trees $\mathcal{T}_w$ and $\mathcal{T}_w^R$.

Let P be an $s - t$ shortest path in $G - e$. We have the following two cases.

1 *P passes through either s' or t':*

If P passes through t', then $d_{G-e}(s, t) = d_{G-e}(s, t') + d(t', t)$, and if P passes through s', then $d_{G-e}(s, t) = d(s, s') + d_{G-e}(s', t)$. So in this case we can use B1 to report the distance between s and t in $G - e$.

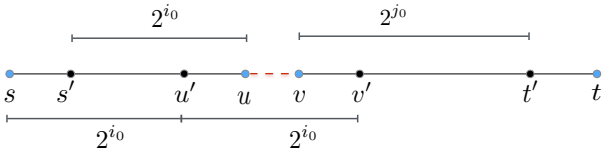


Figure 1 Depiction of vertices s', u', v', t' when the failing edge $e = (u, v)$ lies on path $\pi(s, t)$.

2 P does not pass through s' and t':
Let us assume that $i_0 \leq j_0$. (If $j_0 < i_0$ then a similar analysis will follow). Let u', v' be vertices on $\pi(s, t)$ such that $|\pi(s, u')| = |\pi(u', v')| = 2^{i_0}$. Since $2^{i_0} \leq 2^{j_0}$, we have $u' \in \pi(s', u)$ and $v' \in \pi(v, t')$. Thus P does not pass through segment $\pi(u', v')$, i.e., $\pi_{G-e}(s, t) = \pi_{G-\pi(u',v')}(s, t)$. So in this case, we can use B2 to report $d_{G-e}(s, t)$.

The space and the query time of our data structure are summarized by the following theorem:

▶ **Theorem 2.** *An n-vertex directed or undirected weighted graph G for a given set $W \subseteq V(G)$ can be preprocessed in polynomial time to compute a data structure of $O(n|W| \log n)$ size that given any two vertices $s, t \in W$ and any failing edge e can report $d_{G-e}(s, t)$ in constant time. Our result also holds for single vertex failure.*

3.2 Distance Sensitivity $S \times T$ Oracle

In the following we assume that G is a directed or undirected unweighted graph. Also we assume $|S| \leq |T|$, as otherwise we could consider G^R instead and swap the roles of S and T. Let $L \geq 5$ be a parameter in $[n/\sqrt{|S||T|}, n/\log n]$ and let $R \subseteq V(G)$ be a set of size $O(n/L)$ as obtained from Lemma 1. Also let ℓ be $\lceil L \log n \rceil$. Our construction is a simple two step process:

1. Set $W = S \cup R$, and compute the $W \times W$ oracle of Section 3.1 over set W.
2. For each pair $(s, t) \in S \times T$, if $e_1, e_2, \ldots, e_{\min\{\ell, |\pi(s,t)|\}}$ are the edges on $\pi(s, t)[-\ell]$ listed in reverse order (i.e., from t towards s), then store in $d^{-i}_{(s,t)}$ the distance $d_{G-e_i}(s, t)$.

▶ **Lemma 3.** *Let $e = (u, v)$ be an edge lying on $\pi(s, t)$ for some vertices $s, t \in V(G)$. Also let $x \in V(G)$ be such that $d(x, t) \leq d(v, t)$. Then $e \notin \pi(x, t)$, and so $d_{G-e}(x, t) = d(x, t)$.*

Proof. Let us assume on the contrary that $\pi(x, t)$ traverses e, and let u' (resp. v') be the first (resp. last) vertex in $\{u, v\}$ it encounters. Then

$$d(x, t) = d(x, u') + 1 + d(v', t) \geq 1 + d(v, t) > d(v, t).$$

However, by our hypothesis $d(x, t) \leq d(v, t)$. Hence, we get a contradiction. ◀

▶ **Lemma 4.** *Let $e = (u, v)$ be an edge lying on $\pi(s, t)$ for some vertices $s, t \in V(G)$. Also assume $e \notin \pi(s, t)[-\ell]$, then $d_{G-e}(s, t) = \min_{x \in R,\ d(x,t) \leq \ell} \big(d_{G-e}(s, x) + d(x, t)\big)$.*

Proof. Let P be the shortest path from s to t in $G-e$. Since $|P| \geq d(s, t) > \ell$, by Lemma 1 and sub-optimality of shortest paths, the path $P[-\ell]$ must contain at least one vertex from set R, let this be r. Consider the path $P(r, t) = \pi_{G-e}(r, t)$. Since $d(r, t) \leq |\pi_{G-e}(r, t)| \leq \ell \leq d(v, t)$, Lemma 3 implies that $d_{G-e}(r, t)$ is equal to $d(r, t)$. Therefore we have:

$$d_{G-e}(s, t) = d_{G-e}(s, r) + d_{G-e}(r, t) = d_{G-e}(s, r) + d(r, t).$$

Algorithm 1: Compute $d_{G-e}(s,t)$ where $e=(u,v)$ is a failing edge on $\pi(s,t)$.

1 **if** $(i \leq \ell)$ **then return** $d^{-i}_{(s,t)} = d_{G-e}(s,t)$
2 **else return** $\min_{x\in R,\ d(x,t)\leq \ell} \big(d_{G-e}(s,x)+d(x,t)\big)$

Also notice that for any $r_0 \in R$, if $d(r_0,t) \leq \ell$, then by Lemma 3, $d(r_0,t) = d_{G-e}(r_0,t)$, as $d(v,t) \geq \ell$. Thus $d_{G-e}(s,r_0) + d(r_0,t) = d_{G-e}(s,r_0) + d_{G-e}(r_0,t) \geq d_{G-e}(s,t)$. So from above discussion it follows that $d_{G-e}(s,t) = \min_{x\in R,\ d(x,t)\leq \ell} \big(d_{G-e}(s,x)+d(x,t)\big)$. ◀

Query algorithm. Consider a pair (s,t), let $e=(u,v)$ be a failing edge lying on $\pi(s,t)$. (As before whether e belongs to $\pi(s,t)$ can be verified in constant time). If $\pi(s,t)[-\ell]$ contains e, then we can output new distance in $O(1)$ time. If $\pi(s,t)[-\ell]$ does not contain e, then by Lemma 4, $d_{G-e}(s,t) = \min\{d_{G-e}(s,r) + d(r,t) | r \in R, d(r,t) \leq \ell\}$. In this equation the distance $d_{G-e}(s,r)$ for any $s \in S$ and $r \in R$ can be computed in constant time using the data structure of the previous subsection. Since the values $d(r,t)$ are pre-stored, the query time is $O(|R|) = O(n/L)$. Algorithm 1 presents the pseudocode of our implementation. Notice that the space used is $O(n|W|\log n + |S||T|\ell) = O\big((n^2/L + |S||T|L)\log n\big)$ which, due to our choice of L, is $O(|S||T|L\log n)$. We hence obtain the following result:

▶ **Theorem 5.** *An n-vertex (directed or undirected) unweighted graph G for a given source set $S \subseteq V(G)$ and destination set $T \subseteq V(G)$ can be preprocessed in polynomial time to compute a DSO of size $O(|S||T|L\log n)$ and query time $O(n/L)$, where $L \in [n/\sqrt{|S||T|}, n/\log n]$.*

Notice that the subgraph lower bound of $\Omega(n\sqrt{n|S|})$ provided in [24] holds also for the oracles setting (by using standard information theoretic arguments). Therefore, by choosing $L = \Theta(n/\sqrt{|S||T|})$ in Theorem 5, we obtain an oracle of size $O(n\sqrt{|S||T|}\log n)$ and query time $O(\sqrt{|S||T|})$ which, for $|T| = \Theta(n)$, has optimal size (up to the poly-logarithmic factors).

3.3 Space-Improved $S \times T$ Oracle

Recall that in last subsection we computed an $S \times T$ oracle with $O(n\sqrt{|S||T|}\log n)$ space and $O(\sqrt{|S||T|})$ query time using a random sample of vertices R. In this section, we obtain an oracle with an improved size at the expense of higher (quasi-polynomial) query time. In particular, this oracle has the optimal size for $|T| = \Omega(\sqrt{|S|n})$. Our main idea is to use a hierarchy of random sets $R_1, R_2, \ldots, R_\alpha$ for an appropriate α.

We now explain our construction. Let α be integer to be fixed later on, and for $i \in [0,\alpha]$, let L_i be $(n/|S|)^{\frac{2\alpha-i}{2\alpha}}$ and R_i be random set of size $O(n/L_i) = O(|S|^{\frac{2\alpha-i}{2\alpha}} n^{\frac{i}{2\alpha}})$ computed using Lemma 1. For each $i \geq 0$, we will compute an oracle for $S \times R_i$, say $\mathcal{O}_{S\times R_i}$. Also we use $\mathcal{O}_{S\times T}$ to denote our oracle for the product $S \times T$. Since $|R_0| = O(|S|)$, we use Theorem 2 to compute an oracle for $S \times R_0$ with $O(n|S|\log n)$ space and $O(1)$ query time. It turns out that for any $i > 1$, our oracle $\mathcal{O}_{S\times R_i}$ uses $\mathcal{O}_{S\times R_{i-1}}$, and $\mathcal{O}_{S\times T}$ uses $\mathcal{O}_{S\times R_\alpha}$.

For sake of convenience let $R_{\alpha+1} = T$. For an oracle $\mathcal{O}$, let $\texttt{size}(\mathcal{O})$ be the size of the oracle and let $\texttt{time}(\mathcal{O})$ be its query time. For any $i \in [0,\alpha]$, we compute the $\mathcal{O}_{S\times R_{i+1}}$ oracle from $\mathcal{O}_{S\times R_i}$ as follows. We first compute oracle $\mathcal{O}_{S\times R_i}$, this is augmented by storing

1. $d(x,y)$ for $(x,y) \in R_i \times R_{i+1}$, and
2. $d_{G-e}(s,y)$ for $(s,y) \in S \times R_{i+1}$, $e \in \pi(s,y)[-L_i \log n]$.

So, we have: $\texttt{size}\big(\mathcal{O}_{S\times R_{i+1}}\big) = \texttt{size}\big(\mathcal{O}_{S\times R_i}\big) + (|R_i||R_{i+1}| + |S||R_{i+1}|L_i\log n)$.

For any $y \in R_{i+1}$, to report the distance $d_{G-e}(s,y)$ we proceed in a similar way as in Algorithm 1. If $\pi(s,y)[-L_i \log n]$ contains e we return the stored distance. Otherwise we compute $d_{G-e}(s,y)$ as $\min_{x \in R_i,\ d(x,y) \leq L_i \log n} \big(d_{G-e}(s,x) + d(x,y)\big)$, where $d_{G-e}(s,x)$ is obtained by querying $\mathcal{O}_{S \times R_i}$. Since at most $|R_i|$ queries to $\mathcal{O}_{S \times R_i}$ are performed, we have $\texttt{time}\big(\mathcal{O}_{S \times R_{i+1}}\big) = \texttt{time}\big(\mathcal{O}_{S \times R_i}\big) \times |R_i|$. Summing the first equation from $i = 0$ to α, and substituting $\alpha = \log n - 1$, we obtain the size of oracle $\mathcal{O}_{S \times T}$:

$$\begin{aligned}
\texttt{size}\big(\mathcal{O}_{S \times T}\big) &= \texttt{size}\big(\mathcal{O}_{S \times R_0}\big) + \sum_{i=0}^{\alpha} (|R_i||R_{i+1}| + |S||R_{i+1}|L_i \log n) \\
&\leq n|S| \log n + \alpha |R_\alpha|^2 + |R_\alpha||T| + \sum_{i=0}^{\alpha-1} (|S||R_{i+1}|L_i \log n) + |S||T|L_\alpha \log n \\
&= (\alpha + \log n) n|S| + |T|\sqrt{n|S|} + \sum_{i=0}^{\alpha-1} (n|S|(n/|S|)^{\frac{1}{2\alpha}} \log n) + |T|\sqrt{n|S|} \log n \\
&= O(n|S| \log^2 n + |T|\sqrt{n|S|} \log n).
\end{aligned}$$

Turning to query time, we get that $\texttt{time}(\mathcal{O}_{S \times T}) = O(1) \cdot \prod_{i=0}^{\alpha} |R_i| = (n|S|)^{\frac{\alpha+1}{2}} = O\big((n|S|)^{\frac{\log n}{2}}\big)$. The following theorem follows from above discussion.

▶ **Theorem 6.** *An n-vertex (directed or undirected) unweighted graph G for a given source set $S \subseteq V(G)$ and destination set $T \subseteq V(G)$ can be preprocessed in polynomial time to compute a DSO with $O(n|S| \log^2 n + |T|\sqrt{n|S|} \log n)$ size and $O\big((n|S|)^{\frac{\log n}{2}}\big)$ query time.*

Notice that the subgraph lower bound of $\Omega(n\sqrt{n|S|})$ provided in [24] can be easily adapted to yield an $\Omega(|T|\sqrt{n|S|})$ lower bound for the $S \times T$ case (oracles setting included), and the details will be given in the full version of the paper. Therefore, for $|T| = \Omega(\sqrt{n|S|})$, the oracle of Theorem 6 has optimal size (up to poly-logarithmic factors).

We next show that for the special case of $|T| = \Omega(n^{\frac{3}{4}}|S|^{\frac{1}{4}})$, we can obtain an even better space-optimal oracle (up to logarithmic factors) having *polynomial* query time.

▶ **Theorem 7.** *An n-vertex (directed or undirected) unweighted graph G for a given source set $S \subseteq V(G)$ and destination set $T \subseteq V(G)$ satisfying the condition $|T| = \Omega(n^{\frac{3}{4}}|S|^{\frac{1}{4}})$ can be preprocessed in polynomial time to compute a DSO of size $O\big(T\sqrt{n|S|} \log n\big)$ and query time $O\big(n^{\frac{3}{2}}|S|^{\frac{3}{2}}|T|^{-1}\big) = O(n^{\frac{3}{4}}|S|^{\frac{5}{4}})$.*

Proof. Let L be a parameter and R be a random set of size $O(n/L)$ computed by Lemma 1. Our oracle consists of an $S \times R$ oracle $\mathcal{O}_{S \times R}$ of size $O(|S||R|L_0 \log n)$ and query time $O(n/L_0)$, for some parameter $L_0 \in [n/\sqrt{|S||R|}, n/\log n]$ (see Theorem 5). This is augmented by storing (i) $d(x,t)$ for $x \in R$, $t \in T$, and (ii) the distance $d_{G-e}(s,t)$ for $s \in S$, $t \in T$, $e \in \pi(s,t)[-\ell]$ (recall that $\ell = \lceil L \log n \rceil$). The overall size is $O(|S||R|L_0 \log n + |R||T| + |S||T|L \log n)$.

To report $d_{G-e}(s,t)$ we proceed in a similar way discussed in Algorithm 1. If $e \in \pi(s,t)[-\ell]$ we return the stored distance. Otherwise we compute $d_{G-e}(s,t)$ as $\min_{x \in R,\ d(x,t) \leq \ell} \big(d_{G-e}(s,x) + d(x,t)\big)$, where $d_{G-e}(s,x)$ is obtained by querying $\mathcal{O}_{S \times R}$. Since $O(|R|)$ queries to $\mathcal{O}_{S \times R}$ are performed, the total query time is $O(|R|(n/L_0))$.

Now we get our result by substituting $|R|$ as $\Theta(n/L)$, and picking $L = \sqrt{n/|S|}$ and $L_0 = |T|/|S|$ to obtain an oracle of size $O(T\sqrt{n|S|} \log n)$, and query time $O((n|S|)^{\frac{3}{2}}/|T|)$. ◀

3.4 $S \times T$ Oracle with Additive Distortion

We conclude this section by focusing on the case in which an additive distortion to the reported distance is allowed. The following theorem provides a family of oracles whose additive stretch

decreases as soon as the size increases, regardless of the number of destinations.

▶ **Theorem 8.** *Let $L \in [5, n/|S|]$ and G be an undirected unweighted graph with S as source set and T as destination set, and assume w.l.o.g. that $|S| \leq |T|$. Then, there exists a polynomial-time constructible $(2L \log n)$-additive DSO of $O((n^2/L) \cdot \log n)$ size and $O(1)$ query time.*

For the prominent single-source case, the above result is complemented by the following conditional lower bound:

▶ **Theorem 9.** *Let $\epsilon \in (0, 1]$ be any fixed constant. If the Set-Intersection Conjecture holds, then any single-source DSO with constant query time and additive distortion $d = O(n^{1-\epsilon})$ must use $\tilde{\Omega}\left(n^{\frac{3}{2}\epsilon}\right)$ bits of memory.*

The above result improves several (unconditional) lower bounds on fault-tolerant additive-distortion single-source structures. Moreover, we have recently extended the above lower bound to the case of multiple sources, and we will provide it in the full version of the paper.

4 Path-Reporting $S \times T$ Oracle and Fault-Tolerant Preservers

The following lemma shows that the shortest paths have a very nice structure in the graph $G - e$. (Since all our results in this section will crucially use this lemma, the results in this section will hold for undirected graphs and edge failures only.)

▶ **Lemma 10** (also proved in [21, 10])**.** *Let G be an undirected weighted graph, $s, t \in V(G)$ and $e \in \pi(s, t)$ such that s and t are connected in $G - e$. There exists an edge $(y, z) \in G - e$ such that $\pi(s, y)::(y, z)::\pi(z, t)$ is a shortest path in $G - e$. We will refer to (y, z) by $\textsc{link}(s, t, e)$.*

For $W \times W$, the above lemma implies the following:

▶ **Theorem 11.** *An n-vertex undirected weighted graph G for a given set $W \subseteq V(G)$ can be preprocessed in polynomial time to compute a data structure of $O(n|W| \log n)$ size that given any two vertices $s, t \in W$ and any failing edge $e \in E(G)$, can report $\pi_{G-e}(s, t)$ in $O(|\pi_{G-e}(s, t)|)$ time.*

Moreover, as a by-product we get a sparse subgraph with $O(n|W| \log n)$ edges that preserves distance between any vertex pair $(s, t) \in W \times W$ after single edge failure e.

The above construction can be used to design a path-reporting oracle for $S \times T$, for the unweighted case only though. As before, for a parameter L we take a random set R with $O(n/L)$ vertices. We pre-compute the path reporting oracle for $W \times W$, where $W = S \cup R$, and also the distance oracle for $S \times T$. Recall that this will take $O((n^2/L + |S||T|L) \log n)$ space. Next, for each $(s, t) \in S \times T$ and $e \in \pi(s, t)[-\ell]$, we store the following: (i) edge $(y, z) = \textsc{link}(s, t, e)$, (ii) the distance $d(z, t)$, and (iii) the suffix $\pi_{G-e}(s, t)[-\ell]$.

Notice that the total space used by us is $O((n^2/L) \log n + |S||T|L^2 \log^2 n)$. On choosing L as $n^{2/3}(|S||T|)^{-1/3}$, we get a bound of $O(n^{4/3}(|S||T|)^{1/3} \log^2 n)$. Then, we use the following:

Path-reporting Query Algorithm

1. If $e \notin \pi(s, t)$, we return $\pi(s, t)$ stored in the shortest path tree $\mathcal{T}_s$ (recall $s \in W$).
2. If $e \in \pi(s, t)[-\ell]$ then we retrieve the pre-stored edge $(y, z) = \textsc{link}(s, t, e)$, the distance $d(z, t)$, and the path $P = \pi_{G-e}(s, t)[-\ell]$.
   - If $d(z, t) \leq \ell$, then z must lie on P and $\pi_{G-e}(s, t)$ will be equal to $\pi(s, y)::(y, z)::P(z, t)$.

- If $d(z,t) > \ell$, then we compute a vertex $r \in R$ lying on $P = \pi_{G-e}(s,t)[-\ell]$ in $O(|P|) = O(\ell)$ time. Such an r exists by Lemma 1. We output $\pi_{G-e}(s,t) = \pi_{G-e}(s,r)::\pi(r,t)$. Notice that in this case $\pi_{G-e}(s,t)$ can be outputted in $O(|\pi_{G-e}(s,t)|)$ time.

3. If $e \notin \pi(s,t)[-\ell]$, then it follows from Lemma 4 that $\pi_{G-e}(s,t) = \pi_{G-e}(s,r)::\pi(r,t)$, where $r = \arg\min \big(d_{G-e}(s,x) \ + \ d(x,t) \mid x \in R,\ d(x,t) \leq \ell\big)$. Such an r is computable in $O(|R|) = O(n^{1/3}|S|^{1/3}|T|^{1/3})$ time. Thus in this case in $O(|\pi_{G-e}(s,t)| + (n|S||T|)^{1/3})$ time we can report $\pi_{G-e}(s,t)$.

The above analysis thus implies the following result:

▶ **Theorem 12.** *For any undirected unweighted graph G there exists a polynomial-time constructible DSO for a source set S and destination set T of size $O(n^{4/3}(|S||T|)^{1/3}\log^2 n)$ that for any $(s,t) \in S \times T$, and any failing edge $e \in E(G)$, can report $\pi_{G-e}(s,t)$ in $O(|\pi_{G-e}(s,t)| + (n|S||T|)^{1/3})$ time.*

Moreover, as a by-product we get a sparse subgraph with $O(n^{4/3}|S|^{1/3}|T|^{1/3}\log^2 n)$ edges that preserves distance between any vertex pair $(s,t) \in S \times T$ after single edge failure e.

Finally, we conclude this section by providing an even better oracle for the meaningful scenario in which $T = V(G)$. As before we take a set R with $\sqrt{|S||T|}$ vertices and compute the path reporting oracle for $W \times W$, where $W = S \cup R$. We also pre-compute a distance oracle for $S \times T$. For each $t \in V(G)$, and each $e \in \pi(s,t)[-\ell]$, we store the last edge of $\pi_{G-e}(s,t)$. Notice that the overall size of the oracle remains same, i.e. $O(n\sqrt{|S||T|}\log n)$.

A path query is performed as follows: if $e \notin \pi(s,t)$, we return $\pi(s,t)$ stored in the shortest path tree $\mathcal{T}_s$ (recall $s \in W$); if e appears on $\pi(s,t)[-\ell]$, we can access the last edge, say (w,t), of $P = \pi_{G-e}(s,t)$ in constant time and obtain path $P[s,w] = \pi_{G-e}(s,w)$ by recursively querying the oracle. Finally, in the remaining case, we compute in $O(|R|)$ time the vertex $r = \arg\min\{d_{G-e}(x) + d(z,t) \mid x \in R,\ d(x,t) \leq \ell\}$. We know that the concatenation $\pi_{G-e}(s,r) :: \pi(r,t)$ is a shortest path from s to t in $G - e$. Notice that using Theorem 11, $\pi_{G-e}(s,r)$ can be reported in $O(|\pi_{G-e}(s,r)|)$ time, also the path $\pi(r,t)$ is stored in the tree $\mathcal{T}_r$ (recall $r \in W$). Thus the time for reporting path $\pi_{G-e}(s,t)$ is $O(|R| \ + \ |\pi_{G-e}(s,t)|)$. Notice that $|R| = \sqrt{n|S|}$ and $|\pi_{G-e}(s,t)| \geq d(s,t) \geq \ell = (n/|R|)\log n = \sqrt{n/|S|}\log n$. Therefore in this case as well, the total time spent is of order of the number of edges on the shortest path $\pi_{G-e}(s,t)$. To summarize, we have:

▶ **Theorem 13.** *For any undirected unweighted graph G there exists a polynomial-time constructible DSO for a source set S of size $O(n\sqrt{n|S|}\log n)$ that for any $s \in S$, $t \in V(G)$, and any failing edge $e \in E(G)$, can report $\pi_{G-e}(s,t)$ in $O(|\pi_{G-e}(s,t)|)$ time.*

Remarkably, the above multi-source oracle is the natural counterpart of the multi-source fault-tolerant preserver given in [24], and so it is optimal in space (up to poly-logarithmic factors) and in query time.

5 Distributed Routing Scheme for Single-Source Distances

In this section, we present the main crux of our edge-fault-tolerant routing distributed schemes for the *single-source to all-destinations* case. Details about our labeling scheme and the remaining details of the routing scheme can be found in the full version of the paper. We use s to denote the designated source vertex. For any two vertices a, b, we denote by $\textsc{Treepath}(a,b)$ the path from a to b in tree $\mathcal{T}_s$.

It turns out that finding our labeling scheme of $\tilde{O}(n^{\frac{3}{2}})$ bitsize is quite easy, while designing our routing scheme is not that straightforward. Our approach for it works as follows. First

we show how to represent the FT-BFS (w.r.t. s) subgraph as a union of trees and link-edges (see Lemma 10) of total size $\tilde{O}(n^{3/2})$, so that each replacement path can be represented as a combination of two tree paths connected by a link edge. We can then use the routing scheme over trees by Thorup and Zwick [29]. For ensuring small routing tables, we will need that each vertex must be present in only $\tilde{O}(\sqrt{n})$ number of trees in the family of trees considered by us. In this way, we will obtain a scheme for routing packets from s along replacement shortest paths with poly-logarithmic headers and $\tilde{O}(\sqrt{n})$ bitsize node routing tables and edge labels.

5.1 Tree Representations

For a parameter $L = \sqrt{n}$ we take $R \subseteq V(G)$ to be a set of vertices as obtained from Lemma 1. Also ℓ is taken to be $\lceil L \log n \rceil$. We define two family of trees $\mathcal{T}_{long}$ and $\mathcal{T}_{short}$ as follows: (i) The family $\mathcal{T}_{long}$ consists of shortest path tree $\mathcal{T}_s$, and the shortest path trees $\mathcal{T}_r$ for each $r \in R$; (ii) The family $\mathcal{T}_{short}$ consists of all the possible trees $\mathcal{T}_{e,z}$ given by Definition 14 below and have depth at most ℓ.

▶ **Definition 14.** Let $e = (u, v) \in \mathcal{T}_s$, and (y, z) be an edge that becomes tree edge in $\mathcal{T}_{s,G-e}$. Also assume $d(u, z) \leq 2\ell$. We define $\mathcal{T}_{e,z}$ to be a subtree of $\mathcal{T}_{s,G-e}$ which is (i) rooted at vertex z, and (ii) truncated to depth ℓ, that is, it contains only those vertices whose depth differ from depth of z in $\mathcal{T}_{s,G-e}$ by at most ℓ.

In the following table, we show how the families $\mathcal{T}_{long}$ and $\mathcal{T}_{short}$ can be directly used to obtain a shortest path from s to any arbitrary vertex t after an edge failure on $\textsc{Treepath}(s, t)$.

If	**Then**	**New $s-t$ shortest path in $G-e$**
$t = r_0 \in R$		$\textsc{Treepath}(s, y)::(y, z)::\mathcal{T}_{r_0}(z, r_0)$ where $(y, z) = \textsc{link}(s, r_0, e)$
$e \in \textsc{Treepath}(s, t)[-\ell]$ $(y, z) = \textsc{link}(s, t, e)$, $d(z, t) \leq \ell$	$t \in \mathcal{T}_{e,z} \in \mathcal{T}_{short}$	$\textsc{Treepath}(s, y)::(y, z)::\mathcal{T}_{e,z}(z, t)$
Remaining cases	$\pi_{G-e}(s, t)[-\ell]$ must contain some $r \in R$	$\textsc{Treepath}(s, y_r)::(y_r, z_r)::\mathcal{T}_r(z_r, t)$ where $(y_r, z_r) = \textsc{link}(s, r, e)$

We now show correctness of above table case by case.

Case 1. $t = r_0 \in R$

Let $(y, z) = \textsc{link}(s, r_0, e)$. Recall that we showed in Lemma 10, $\pi(s, y)::(y, z)::\pi(z, r_0) = \textsc{Treepath}(s, y)::(y, z)::\mathcal{T}_{r_0}(z, r_0)$ is a shortest path from s to r_0 in $G - e$. Notice that the trees $\mathcal{T}_s$ and $\mathcal{T}_{r_0}$ are present in the family $\mathcal{T}_{long}$.

Case 2. $e \in \textsc{Treepath}(s, t)[-\ell]$, $(y, z) = \textsc{link}(s, t, e)$, $d(z, t) \leq \ell$

Since $d(t, z) \leq \ell$, we will have $d(u, z) \leq 2\ell$. This along with the fact that (y, z) becomes a tree edge in $\mathcal{T}_{s,G-e}$ shows that tree $\mathcal{T}_{e,z}$ is present in the family $\mathcal{T}_{short}$. Also from Lemma 10 we know that e cannot lie on $\pi(z, t)$, so $d_{G-e}(z, t) = d(z, t) \leq \ell$. Since tree $\mathcal{T}_{e,z}$ contains shortest paths up to depth ℓ, vertex t must lie in $\mathcal{T}_{e,z}$. This shows that $\mathcal{T}_{e,z}(z, t) = \pi_{G-e}(z, t) = \pi(z, t)$. So by applying Lemma 10, we get that $\textsc{Treepath}(s, y)::(y, z)::\mathcal{T}_{e,z}(z, t)$ is a shortest path from s to t in $G - e$.

Case 3(i). $e \in \textsc{Treepath}(s, t)[-\ell]$, $(y, z) = \textsc{link}(s, t, e)$, $d(z, t) > \ell$

Let $P = \pi(s, y)::(y, z)::\pi(z, t)$ be a shortest path from s to t in $G \setminus e$ (see Lemma 10). As $|P(z, t)| \geq \ell$, by Lemma 1, $P[-\ell]$ must contain a vertex from set R, say r. Let $(y_r, z_r) = \textsc{link}(s, r, e)$. Since $P[s, r] = \pi_{G-e}(s, r)$, edge (y_r, z_r) must be identical to the

edge (y, z). Notice that $\pi(z,t) = \mathcal{T}_r(z,r)::\mathcal{T}_r(r,t) = \mathcal{T}_r(z,t) = \mathcal{T}_r(z_r,t)$. Thus in this case $\textsc{Treepath}(s,y_r)::(y_r,z_r)::\mathcal{T}_r(z_r,t)$ is a shortest path from s to t in the graph $G-e$.

Case 3(ii). $e \notin \textsc{Treepath}(s,t)[-\ell]$

Let $P = \pi_{G-e}(s,t)$. By Lemma 1, $P[-\ell]$ must contain a vertex from set R, say r. We know by Lemma 3 that $P[r,t]$ is a shortest path in G, thus $P[r,t] = \mathcal{T}_r(r,t)$. Let $(y_r,z_r) = \textsc{link}(s,r,e)$, then $P[s,r] = \pi_{G-e}(s,r) = \textsc{Treepath}(s,y_r)::(y_r,z_r)::\mathcal{T}_r(z_r,r)$. Thus in this case also $\pi_{G-e}(s,t) = \textsc{Treepath}(s,y_r)::(y_r,z_r)::\mathcal{T}_r(z_r,t)$. (Notice that if $\textsc{Treepath}(s,r)$ is intact in graph $G-e$, then we can define $\textsc{link}(s,r,e)$ to be any arbitrary edge on $\textsc{Treepath}(s,r)$).

All that remains is to show that each vertex in G appears in $\tilde{O}(\sqrt{n})$ trees in the families $\mathcal{T}_{short}$ and $\mathcal{T}_{long}$. For the family $\mathcal{T}_{long}$, proof is trivial because $|\mathcal{T}_{long}| = O(|R|) = O(\sqrt{n})$, and it turns out the same holds for the family $\mathcal{T}_{short}$. From this, the bound on the size of our routing scheme will follow.

References

1. A. Abboud and G. Bodwin. The 4/3 additive spanner exponent is tight. In *STOC*, pages 351–361, 2016.
2. Ittai Abraham, Shiri Chechik, Cyril Gavoille, and David Peleg. Forbidden-set distance labels for graphs of bounded doubling dimension. *ACM Transactions on Algorithms (TALG)*, 12(2):22, 2016.
3. Surender Baswana and Neelesh Khanna. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica*, 66(1):18–50, 2013.
4. Michael A. Bender and Martin Farach-Colton. The level ancestor problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004. doi:10.1016/j.tcs.2003.05.002.
5. Aaron Bernstein and David R. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 101–110, 2009.
6. Davide Bilò, Fabrizio Grandoni, Luciano Gualà, Stefano Leucci, and Guido Proietti. Improved purely additive fault-tolerant spanners. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 167–178, 2015.
7. Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Fault-tolerant approximate shortest-path trees. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 137–148, 2014.
8. Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Compact and fast sensitivity oracles for single-source distances. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 13:1–13:14, 2016.
9. Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 18:1–18:14, 2016.
10. Greg Bodwin, Fabrizio Grandoni, Merav Parter, and Virginia Vassilevska Williams. Preserving Distances in Very Faulty Graphs. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 73:1–73:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

11 Jean Bourgain. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985.

12 Shiri Chechik. Fault-tolerant compact routing schemes for general graphs. *Inf. Comput.*, 222:36–44, 2013.

13 Shiri Chechik, Sarel Cohen, Amos Fiat, and Haim Kaplan. (1 + epsilon)-approximate f-sensitive distance oracles. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1479–1496, 2017.

14 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f-sensitivity distance oracles and routing schemes. *Algorithmica*, 63(4):861–882, 2012.

15 Don Coppersmith and Michael Elkin. Sparse source-wise and pair-wise distance preservers. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 660–669. SIAM, 2005. URL: `http://dl.acm.org/citation.cfm?id=1070432.1070524`.

16 Bruno Courcelle and Andrew Twigg. Compact forbidden-set routing. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 37–48. Springer, 2007.

17 Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.

18 Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *SODA'09: Proceedings of 19th Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 506–515, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

19 Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 748–757, 2012.

20 Telikepalli Kavitha. New pairwise spanners. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 513–526. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. `doi:10.4230/LIPIcs.STACS.2015.513`.

21 Kavindra Malik, A.K. Mittal, and Sumit K. Gupta. The k most vital arcs in the shortest path problem. *Oper. Res. Lett.*, 8:223–227, 1989.

22 Jiří Matoušek. On the distortion required for embedding finite metric spaces into normed spaces. *Israel Journal of Mathematics*, 93(1):333–344, 1996.

23 Merav Parter. Vertex fault tolerant additive spanners. In *International Symposium on Distributed Computing*, pages 167–181. Springer, 2014.

24 Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 779–790, 2013.

25 Merav Parter and David Peleg. Fault tolerant approximate BFS structures. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1073–1092, 2014.

26 Mihai Patrascu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 815–823. IEEE, 2010.

27 Mihai Patrascu, Liam Roditty, and Mikkel Thorup. A new infinity of distance oracles for sparse graphs. In *Foundations of Computer Science (focs), 2012 Ieee 53rd Annual Symposium on*, pages 738–747. IEEE, 2012.

28 Christian Sommer, Elad Verbin, and Wei Yu. Distance oracles for sparse graphs. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 703–712. IEEE, 2009.

29 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *SPAA*, pages 1–10, 2001.

30 Oren Weimann and Raphael Yuster. Replacement paths via fast matrix multiplication. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 655–662, 2010.

On the Tree Conjecture for the Network Creation Game

Davide Bilò
Department of Humanities and Social Sciences, University of Sassari, Italy
davidebilo@uniss.it

Pascal Lenzner
Algorithm Engineering Group, Hasso Plattner Institute Potsdam, Germany
pascal.lenzner@hpi.de

Abstract

Selfish Network Creation focuses on modeling real world networks from a game-theoretic point of view. One of the classic models by Fabrikant et al. [PODC'03] is the network creation game, where agents correspond to nodes in a network which buy incident edges for the price of α per edge to minimize their total distance to all other nodes. The model is well-studied but still has intriguing open problems. The most famous conjectures state that the price of anarchy is constant for all α and that for $\alpha \geq n$ all equilibrium networks are trees.

We introduce a novel technique for analyzing stable networks for high edge-price α and employ it to improve on the best known bounds for both conjectures. In particular we show that for $\alpha > 4n - 13$ all equilibrium networks must be trees, which implies a constant price of anarchy for this range of α. Moreover, we also improve the constant upper bound on the price of anarchy for equilibrium trees.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory, Theory of computation → Quality of equilibria, Theory of computation → Network formation

Keywords and phrases Algorithmic Game Theory, Network Creation Game, Price of Anarchy, Quality of Nash Equilibria

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.14

Related Version See `http://arxiv.org/abs/1710.01782`, [14] for the full version of the paper.

Acknowledgements We thank our anonymous reviewers for their helpful comments.

1 Introduction

Many important networks, e.g. the Internet or social networks, have been created in a decentralized way by selfishly acting agents. Modeling and understanding such networks is an important challenge for researchers in the fields of Computer Science, Network Science, Economics and Social Sciences. A significant part of this research focuses on assessing the impact of the agents' selfish behavior on the overall network quality measured by the *price of anarchy* [32]. Clearly, if there is no or little coordination among the egoistic agents, then it cannot be expected that the obtained networks minimize the social cost. The reason for this is that each agent aims to improve the network quality for herself while minimizing the spent cost. However, empirical observations, e.g. the famous small-world phenomenon [40, 30, 9], suggest that selfishly built networks are indeed very efficient in terms of the overall cost and of the individually perceived service quality. Thus, it is a main challenge to justify these observations analytically.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 14; pp. 14:1–14:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


A very promising approach towards this justification is to model the creation of a network as a strategic game which yields networks as equilibrium outcomes and then to investigate the quality of these networks. For this, a thorough understanding of the structural properties of such equilibrium networks is the key.

We contribute to this endeavor by providing new insights into the structure of equilibrium networks for one of the classical models of selfish network creation [24]. In this model, agents correspond to nodes in a network and can buy costly links to other nodes to minimize their total distance in the created network. Our insights yield improved bounds on the price of anarchy and significant progress towards settling the so-called tree conjecture [24, 36].

1.1 Model and Definitions

We consider the classical *network creation game* as introduced by Fabrikant et al. [24]. There are n agents V, which correspond to nodes in a network, who want to create a connected network among themselves. Each agent selfishly strives for minimizing her cost for creating network links while maximizing her own connection quality. All edges in the network are undirected and unweighted and agents can create any incident edge for the price of $\alpha > 0$, where α is a fixed parameter of the game. The strategy $S_u \subseteq V \setminus \{u\}$ of an agent u denotes which edges are bought by this agent, that is, agent u is willing to create (and pay for) all the edges (u, x), for all $x \in S_u$. Let $\mathbf{s}$ be the n-dimensional vector of the strategies of all agents. The strategy-vector $\mathbf{s}$ induces an undirected network $G(\mathbf{s}) = (V, E(\mathbf{s}))$, where for each edge $(u, v) \in E(\mathbf{s})$ we have $v \in S_u$ or $u \in S_v$. If $v \in S_u$, then we say that agent u is the *owner* of edge (u, v) or that agent u *buys* the edge (u, v), otherwise, if $u \in S_v$, then agent v owns or buys the edge (u, v).[1] Since the created networks will heavily depend on α we emphasize this by writing $(G(\mathbf{s}), \alpha)$ instead of $G(\mathbf{s})$. The *cost* of an agent u in the network $(G(\mathbf{s}), \alpha)$ is the sum of her cost for buying edges, called the *creation cost*, and her cost for using the network, called the *distance cost*, which depends on agent u's distances to all other nodes within the network. The cost of u is defined as

$$\texttt{cost}(G(\mathbf{s}), \alpha, u) = \alpha|S_u| + \texttt{distcost}(G(\mathbf{s}), u),$$

where the distance cost is defined as

$$\texttt{distcost}(G(\mathbf{s}), u) = \begin{cases} \sum_{w \in V} d_{G(\mathbf{s})}(u, w), & \text{if } G(\mathbf{s}) \text{ is connected} \\ \infty, & \text{otherwise.} \end{cases}$$

Here $d_{G(\mathbf{s})}(u, w)$ denotes the length of a shortest path between u and w in the network $G(\mathbf{s})$. We will mostly omit the reference to the strategy vector, since it is clear that a strategy vector directly induces a network and vice versa. Moreover, if the network is clear from the context, then we will also omit the reference to the network, e.g. writing $\texttt{distcost}(u)$ instead of $\texttt{distcost}(G, u)$.

A network $(G(\mathbf{s}), \alpha)$ is in *pure Nash equilibrium* (NE), if no agent can unilaterally change her strategy to strictly decrease her cost. That is, in a NE network no agent can profit by a strategy change if all other agents stick to their strategies. Since in a NE network no agent wants to change the network, we call them *stable*.

The *social cost*, denoted $\texttt{cost}(G(\mathbf{s}), \alpha)$, of a network $(G(\mathbf{s}), \alpha)$ is the sum of the cost of all agents, that is, $\texttt{cost}(G(\mathbf{s}), \alpha) = \sum_{u \in V} \texttt{cost}(G(\mathbf{s}), \alpha, u)$. Let OPT_n be the minimum social

[1] No edge can have two owners in any equilibrium network. Hence, we will assume throughout the paper that each edge in $E(\mathbf{s})$ has a unique owner.

cost of a n agent network and let maxNE_n be the maximum social cost of any NE network on n agents. The *price of anarchy* (PoA) [32] is the maximum over all n of the ratio $\frac{\text{maxNE}_n}{\text{OPT}_n}$.

Let $G = (V, E)$ be any undirected connected graph with n vertices. A *cut-vertex* x of G is a vertex with the property that G with vertex x removed contains at least two connected components. We say that G is *biconnected* if $n \geq 3$ and G contains no cut-vertex. A *biconnected component* H of G is a maximal induced subgraph of G which is also biconnected. Note that we rule out trivial biconnected components which contain exactly one edge. Thus, there exist at least two vertex-disjoint paths between any pair of vertices x, y in a biconnected component H, which implies that there exists a simple cycle containing x and y.

1.2 Related Work

Network creation games, as defined above, were introduced by Fabrikant et al. [24]. They gave the first general bound of $\mathcal{O}(\sqrt{\alpha})$ on the PoA and they conjectured that above some constant edge-price all NE networks are trees. This conjecture, called the *tree conjecture*, is especially interesting since they also showed that tree networks in NE have constant PoA. In particular, they proved that the PoA of stable tree networks is at most 5. Interestingly, the tree conjecture in its general version was later disproved by Albers et al. [1]. However, non-tree NE networks are known only when $\alpha < n$, in particular, for every $\varepsilon > 0$, there exist non-tree NE networks with $\alpha \leq n - \varepsilon$ [36]. It is believed that for $\alpha \geq n$ the tree conjecture may be true. Settling this claim is currently a major open problem and there has been a series of papers which improved bounds concerning the tree conjecture.

First, Albers et al. [1] proved that for $\alpha \geq 12n \log n$ every NE network is a tree. Then, using a technique based on the average degree of the biconnected component, this was significantly improved to $\alpha > 273n$ by Mihalák & Schlegel [39] and even further to $\alpha \geq 65n$ by Mamageishvili et al. [36]. The main idea of this average degree technique is to prove a lower and an upper bound on the average degree of the unique biconnected component in any equilibrium network. The lower bound has the form "for $\alpha > c_1 n$ the average degree is at least c_2" and the upper bound has the form "for $\alpha > c_3 n$ the average degree is at most $f(\alpha)$", where c_1, c_2, c_3 are constants and f is a function which monotonically decreases in α. For large enough α both bounds contradict each other, which proves that equilibrium networks for this α cannot have a biconnected component and thus must be trees. Very recently a preprint by Àlvarez & Messegué [5] was announced which invokes the average degree technique with a stronger lower bound. This then yields a contradiction already for $\alpha > 17n$. For their stronger lower bound the authors use that in every minimal cycle (we call them "min cycles") of an equilibrium network all agents in the cycle buy exactly one edge of the cycle. This fact has been independently established by us [35] and we also use it.

The currently best general upper bound of $2^{\mathcal{O}(\sqrt{\log n})}$ on the PoA is due to Demaine et al. [22] and it is known that the PoA is constant if $\alpha < n^{1-\epsilon}$ for any fixed $\epsilon \geq \frac{1}{\log n}$ [22]. Thus, the PoA was shown to be constant for almost all α, except for the range between $n^{1-\epsilon}$, for any fixed $\epsilon \geq \frac{1}{\log n}$, and $\alpha < 65n$ (or $\alpha \leq 9n$ which is claimed in [5]). It is widely conjectured that the PoA is constant for all α and settling this open question is a long standing problem in the field. A constant PoA proves that agents create socially close-to-optimal networks even without central coordination. Quite recently, a constant PoA was proven by Chauhan et al. [16] for a version with non-uniform edge prices. In contrast, non-constant lower bounds on the PoA have been proven for local versions of the network creation game by Bilò et al. [10, 12] and Cord-Landwehr & Lenzner [19].

For other variants and aspects of network creation games, we refer the reader to [26, 8, 18, 15, 21, 6, 31, 33, 34, 38, 29, 2, 37, 20, 23, 13, 11, 7, 25, 17, 3, 4, 27].

1.3 Our Contribution

In this paper we introduce a new technique for analyzing stable non-tree networks for high edge-price α and use it to improve on the current best lower bound for α for which all stable networks must be trees. In particular, we prove that for $\alpha > 4n - 13$ any stable network must be a tree (see Section 2). This is a significant improvement over the known bound of $\alpha > 65n$ by Mamageishvili et al. [36] and the recently claimed bound of $\alpha > 17n$ by Àlvarez & Messegué [5]. Since the price of anarchy for stable tree networks is constant [24], our bound directly implies a constant price of anarchy for $\alpha > 4n - 13$. Moreover, in Section 3, we also give a refined analysis of the price of anarchy of stable tree networks and thereby improve the best known constant upper bound for stable trees.

Thus, we make significant progress towards settling the tree conjecture in network creation games and we enlarge the range of α for which the price of anarchy is provably constant.

Our new technique exploits properties of cycles in stable networks by focusing on *critical pairs*, *strong critical pairs* and *min cycles*. The latter have been introduced in our earlier work [35] and are also used in the preprint by Àlvarez & Messegué [5]. However, in contrast to the last attempts for settling the tree conjecture [39, 36, 5], we do not rely on the average degree technique. Instead we propose a more direct and entirely structural approach using (strong) critical pairs in combination with min cycles. Besides giving better bounds with a simpler technique, we believe that this approach is better suited for finally resolving the tree conjecture.

Due to space constraints, all omitted details can be found in the full version [14].

2 Improving the Range of α of the Tree Conjecture

In this section we prove our main result, that is, we show that for $\alpha > 4n - 13$, every NE network (G, α) with $n \geq 4$ nodes must be a tree.

We proceed by first establishing properties of cycles in stable networks. Then we introduce the key concepts called critical pairs, strong critical pairs and min cycles. Finally, we provide the last ingredient, which is a critical pair with a specific additional property, and combine all ingredients to obtain the claimed result.

2.1 Properties of Cycles in Stable Networks

We begin by showing that for large values of α, stable networks cannot contain cycles of length either 3 or 4.

▶ **Lemma 1.** *For $\alpha > \frac{n-1}{2}$, no stable network (G, α) contains a cycle of length 3.*

Proof. Let (G, α) be a stable network for a fixed value of $\alpha > \frac{n-1}{2}$. For the sake of contradiction, assume that G contains a cycle C of length 3. Assume that $V(C) = \{u_0, u_1, u_2\}$ and that C contains the three edges (u_0, u_1), (u_1, u_2), and (u_2, u_0). Let $V_i = \{x \in V \mid d_G(u_i, x) < d_G(u_j, x), \forall j \neq i\}$. Observe that, for every $i \in \{0, 1, 2\}$ we have $|V_i| \geq 1$, as $u_i \in V_i$. Furthermore, all the V_i's are pairwise disjoint. W.l.o.g., assume that $|V_0| = \max\{|V_0|, |V_1|, |V_2|\}$. Furthermore, w.l.o.g., assume that u_1 buys the edge (u_1, u_2). Consider the strategy change in which agent u_1 deletes the edge (u_1, u_2). The building cost of the agent decreases by α while her distance cost increases by at most $|V_2|$. Since $|V_2| \leq |V_0|$, from $|V_0| + |V_1| + |V_2| \leq n$ we obtain $|V_2| \leq \frac{n-1}{2}$. Since G is stable, $\frac{n-1}{2} - \alpha \geq 0$, i.e., $\alpha \leq \frac{n-1}{2}$, a contradiction. ◀

▶ **Lemma 2.** *For $\alpha > n - 2$, no stable network (G, α) contains a cycle of length 4.*

Proof. Let (G,α) be a stable network for a fixed value of $\alpha > n-2$. For the sake of contradiction, assume that G contains a cycle C of length 4. Assume that $V(C) = \{u_0, u_1, u_2, u_3\}$ and that C contains the four edges (u_0,u_1), (u_1,u_2), (u_2,u_3), and (u_3,u_0). For the rest of this proof, we assume that all indices are modulo 4 in order to simplify notation. Let $V_i = \{x \in V \mid d_G(u_i,x) < d_G(u_j,x), \forall j \neq i\}$. Observe that for every $i \in \{0,1,2,3\}$ we have $|V_i| \geq 1$, as $u_i \in V_i$. Let $Z_i = \{x \in V \mid d_G(u_i,x) = d_G(u_{i-1},x)$ and $d_G(u_i,x), d_G(u_{i-1},x) < d_G(u_j,x), \forall j \neq i, i-1\}$. Observe that in the families of the sets V_i and Z_i every pair of sets is pairwise disjoint.

We now rule out the case in which an agent owns two edges of C. W.l.o.g., assume that agent u_0 owns the two edges (u_0,u_1) and (u_0,u_3). Consider the strategy change in which agent u_0 swaps[2] the edge (u_0,u_1) with the edge (u_0,u_2) and, at the same time, deletes the edge (u_0,u_3). The creation cost of agent u_0 decreases by α, while her distance cost increases by $|V_1| + |V_3| - |V_2|$. Since (G,α) is stable, agent u_0 has no incentive in deviating from her current strategy. Therefore, $|V_1| + |V_3| - |V_2| - \alpha \geq 0$, i.e., $\alpha \leq |V_1| + |V_3| - |V_2| \leq n - |V_0| - |V_2| - |V_2| \leq n-3$, where the last but one inequality follows from the pairwise disjointness of all V_i sets, which implies $|V_0| + |V_1| + |V_2| + |V_3| \leq n$. Since, $\alpha > n-2$, no agent can own two edges of C. Therefore, to prove the claim, we need to show that no agent can own a single edge of C.

W.l.o.g., assume that for every $i \in \{0,1,2,3\}$ agent u_i owns the edge (u_i,u_{i+1}). Moreover, w.l.o.g., assume that $|V_1| + |Z_2| = \min_{0 \leq i \leq 3} \{|V_i| + |Z_{i+1}|\}$. Since $\sum_{0 \leq i \leq 3} (|V_i| + |Z_{i+1}|) \leq n$, we have that $|V_1| + |Z_2| \leq \frac{n}{4}$.

Consider the strategy change in which agent u_0 deletes the edge (u_0,u_1). The creation cost of agent u_0 decreases by α, while her distance cost increases by $2|V_1| + |Z_2| \leq \frac{n}{2}$.

Since (G,α) is stable, agent u_0 has no incentive to deviate from her current strategy. Therefore, $\frac{n}{2} - \alpha \geq 0$, i.e., $\alpha \leq \frac{n}{2} \leq n-2$, when $n \geq 4$. We have obtained a contradiction. ◀

▶ **Definition 3** (Directed Cycle). Let C be a cycle of (G,α) of length k. We say that C is *directed* if there is an ordering $u_0, \ldots, u_{k-1}$ of its k vertices such that, for every $i = 0, \ldots, k-1$, $(u_i, u_{(i+1) \mod k})$ is an edge of C which is bought by agent u_i.

We now show that if α is large enough, then directed cycles cannot be contained in a stable network as a biconnected component.

▶ **Lemma 4.** *For $\alpha > n-2$, no stable network (G,α) with $n \geq 6$ vertices contains a biconnected component which is also a directed cycle.*

Proof. Let (G,α) be a stable network for a fixed value of $\alpha > n-2$. Let H be a biconnected component of G. For the sake of contradiction, assume that H is a directed cycle of length k. We can apply Lemma 2 to exclude the case in which $k = 4$. Similarly, since $\alpha > n-2 \geq \frac{n-1}{2}$ for every $n \geq 3$, we can use Lemma 1 to exclude the case in which $k = 3$.

Let $u_0, \ldots, u_{k-1}$ be the k vertices of H and, w.l.o.g., assume that every agent u_i is buying an edge towards agent $u_{(i+1) \mod k}$. To simplify notation, in the rest of this proof we assume that all indices are modulo k. Let $V_i = \{x \in V \mid d_G(u_i,x) < d_G(u_j,x), \forall j \neq i\}$. Observe that V_i is a partition of V. We divide the proof into two cases.

The first case occurs when H is a cycle of length $k \geq 6$. W.l.o.g., assume that $|V_2| = \max_{0 \leq i \leq k-1} |V_i|$. In this case, consider the strategy change of agent u_0 when she swaps the

[2] A *swap* of edge (a,b) to edge (a,c) by agent a who owns edge (a,b) consists of deleting edge (a,b) and buying edge (a,c).

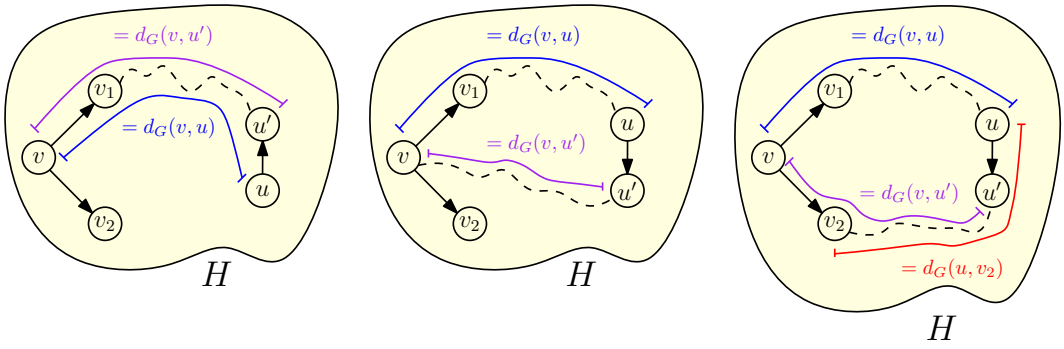


Figure 1 Illustrations of a critical pair $\langle v, u \rangle$. Edge-ownership is depicted by directing edges away from their owner. **Left:** Edge (u, u') belongs to a shortest path tree T rooted at v and u' is the parent of u in T. **Middle:** Edge (u, u') does not belong to any shortest path tree T rooted at v. Note that in this case (v, v_2) can also be on the shortest path from v to u'. **Right:** Illustration of a strong critical pair $\langle v, u \rangle$.

edge (u_0, u_1) with the edge (u_0, u_2). The distance cost of agent u_0 increases by $|V_1| - |V_2| - |V_3| \leq -1$. Thus, agent u_0 has an improving strategy, a contradiction.

The second and last case occurs when H is a cycle of length $k = 5$. If $|V_i| \neq |V_j|$ for some $i, j \in \{0, 1, 2, 3, 4\}$, then there exists an $i \in \{0, 1, 2, 3, 4\}$ such that $|V_i| < |V_{i+1}|$. W.l.o.g., let $|V_1| < |V_2|$. The distance cost of agent u_0 when she swaps the edge (u_0, u_1) with the edge (u_0, u_2) increases by $|V_1| - |V_2| \leq -1$. Thus agent u_0 has an improving strategy, a contradiction. If $|V_0| = |V_1| = |V_2| = |V_3| = |V_4|$, then the increase in the overall cost incurred by agent u_0 when she deletes the edge (u_0, u_1) would be equal to $3|V_1| + |V_2| - \alpha = \frac{4}{5}n - \alpha$. Since G is stable and n is a multiple of 5, $\frac{4}{5}n - \alpha \geq 0$, i.e., $\alpha \leq \frac{4}{5}n \leq n - 2$, for every $n \geq 10$, a contradiction. ◀

2.2 Critical Pairs

The next definition introduces the concept of a (strong) critical pair. As we will see, (strong) critical pairs are the first key ingredient for our analysis. Essentially, we will show that stable networks cannot have critical pairs, if α is large enough.

▶ **Definition 5** (Critical Pair)**.** Let (G, α) be a non-tree network and let H be a biconnected component of G. We say that $\langle v, u \rangle$ is a *critical pair* if all of the following five properties hold:

1. Agent $v \in V(H)$ buys two distinct non-bridge edges, say (v, v_1) and (v, v_2), with $v_1, v_2 \in V(H)$;
2. Agent $u \in V(H)$, with $u \neq v$ buys at least one edge (u, u') with $u' \in V(H)$ and $u' \neq v$;
3. $d_G(v, u) \geq 2$;
4. there is a shortest path between v and u in G which uses the edge (v, v_1);
5. there is a shortest path between v and u' in G which does not use the edge (u, u').

The critical pair $\langle v, u \rangle$ is *strong* if there is a shortest path between u and v_2 which does not use the edge (v, v_2). See Fig. 1 for an illustration.

In the rest of this section, when we say that two vertices v and u of G form a critical pair, we will denote by v_1, v_2, and u' the vertices corresponding to the critical pair $\langle v, u \rangle$ that satisfy all the conditions given in Definition 5. We can observe the following.

▶ **Observation 6.** *If $\langle v, u\rangle$ is a critical pair of a network (G, α), then there exists a shortest path tree T of G rooted at v, where either the edge (u, u') is not an edge of T or u' is the parent of u in T.*

▶ **Observation 7.** *If $\langle v, u\rangle$ is a critical pair, then for every shortest path tree T of (G, α) rooted at u, either the edge (v, v_1) is not an edge of T or v_1 is the parent of v in T. Furthermore, if $\langle v, u\rangle$ is a strong critical pair, then there is a shortest path tree of G rooted at u such that the edge (v, v_2) is not contained in the shortest path tree.*

The next technical lemma provides useful bounds on the distance cost of the nodes involved in a critical pair.

▶ **Lemma 8.** *Let (G, α) be a stable network and let a, b be two distinct vertices of G such that a buys an edge (a, a'), with $a' \neq b$. If $d_G(a, b) \geq 2$ and there exists a shortest path tree T of G rooted at b such that either (a, a') is not an edge of T or a' is the parent of a in T, then $\texttt{distcost}(a) \leq \texttt{distcost}(b) + n - 3$. Furthermore, if a is buying also the edge (a, a''), with $a'' \neq a'$, $a'' \neq b$, and (a, a'') is not an edge of T, then $\texttt{distcost}(a) \leq \texttt{distcost}(b) + n - 3 - \alpha$.*

Proof. Consider the strategy change in which agent a swaps the edge (a, a') with the edge (a, b) and deletes any other edge she owns and which is not contained in T, if any. Let T' be a shortest path tree rooted at b of the graph obtained after the swap. Observe that $d_{T'}(b, x) \leq d_G(b, x)$, for every $x \in V$. Furthermore, as $d_G(a, b) \geq 2$, while $d_{T'}(a, b) = 1$, we have $d_{T'}(a, b) \leq d_G(a, b) - 1$. Therefore, $\sum_{x \in V} d_{T'}(b, x) \leq \texttt{distcost}(b) - 1$. Moreover, the distance from a to every $x \neq a$ is at most $1 + d_{T'}(b, x)$. Finally, the distance from a to herself, which is clearly 0, is exactly 1 less than the distance from b to a in T'. Therefore the distance cost of a in T' is less than or equal to $\texttt{distcost}(b) - 1 + (n - 1) - 1 = \texttt{distcost}(b) + n - 3$.

If besides performing the mentioned swap agent a additionally saves at least α in cost by deleting at least one additional edge which is not in T, then $\texttt{distcost}(a) \leq \texttt{distcost}(b) + n - 3 - \alpha$. This is true since G is stable, which implies that the overall cost of a in G cannot be larger than the overall cost of a after the strategy change. ◀

Now we employ Lemma 8 to prove the structural property that stable networks cannot contain strong critical pairs if α is large enough.

▶ **Lemma 9.** *For $\alpha > 2n - 6$, no stable network (G, α) contains a strong critical pair.*

Proof. Let (G, α) be a non-tree stable network for a fixed value of $\alpha > 2n - 6$ and, for the sake of contradiction, let $\langle v, u\rangle$ be a strong critical pair. Using Observation 6 together with Lemma 8 (where $a = u, a' = u'$, and $b = v$), we have that

$$\texttt{distcost}(u) \leq \texttt{distcost}(v) + n - 3.$$

Furthermore, using Observation 7 together with Lemma 8 (where $a = v, a' = v_1, a'' = v_2$, and $b = u$), we have that

$$\texttt{distcost}(v) \leq \texttt{distcost}(u) + n - 3 - \alpha.$$

By summing up both the left-hand and the right-hand side of the two inequalities we obtain $0 \leq 2n - 6 - \alpha$, i.e., $\alpha \leq 2n - 6$, a contradiction. ◀

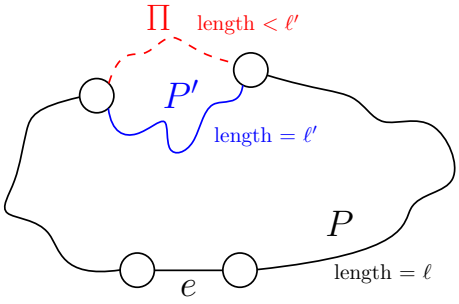


Figure 2 The cycle C containing edge e and the paths P, P' and Π.

2.3 Min Cycles

We now introduce the second key ingredient for our analysis: min cycles.

▶ **Definition 10** (Min Cycle)**.** Let (G, α) be a non-tree network and let C be a cycle in G. We say that C is a min cycle if, for every two vertices $x, x' \in V(C)$, $d_C(x, x') = d_G(x, x')$.

First, we show that every edge of every biconnected graph is contained in some min cycle. This was also proven in [35] and [5].

▶ **Lemma 11.** *Let H be a biconnected graph. Then, for every edge e of H, there is a min cycle that contains the edge e.*

Proof. Since H is biconnected, there exists at least a cycle containing the edge e. Among all the cycles in H that contain the edge e, let C be a cycle of minimum length. We claim that C is a min cycle. For the sake of contradiction, assume that C is not a min cycle. This implies that there are two vertices $x, y \in V(C)$ such that $d_H(x, y) < d_C(x, y)$. Among all pairs $x, y \in V(C)$ of vertices such that $d_H(x, y) < d_C(x, y)$, let x', y' be the one that minimizes the value $d_H(x', y')$ (ties are broken arbitrarily). Let Π be a shortest path between x' and y' in G. By the choice of x' and y', Π is edge disjoint from C. Let P and P' be the two edge-disjoint paths between x' and y' in C and, w.l.o.g., assume that e is contained in P. Let ℓ and ℓ' be the length of P and P', respectively. See Fig. 2. Clearly, the length of C is equal to $\ell + \ell'$. Since $d_C(x', y') \leq \ell'$, we obtain $d_H(x', y') < \ell'$. Therefore, the cycle obtained by concatenating P and Π has a length equal to $\ell + d_H(x', y') < \ell + \ell'$, and therefore, it is strictly shorter than C, a contradiction. ◀

Now we proceed with showing that stable networks contain only min cycles which are directed and not too short. For this, we employ our knowledge about strong critical pairs.

▶ **Lemma 12.** *For $\alpha > 2n - 6$, every min cycle of a non-tree stable network (G, α) with $n \geq 4$ vertices is directed and has a length of at least 5.*

Proof. Let (G, α) be a non-tree stable network for a fixed $\alpha > 2n - 6$ and let C be a min cycle of G. Since $2n - 6 \geq \frac{n-1}{2}$ for every $n \geq 4$, using Lemma 1, we have that C cannot be a cycle of length equal to 3. Furthermore, Since $2n - 6 \geq n - 2$ for every $n \geq 4$, using Lemma 2, we have that C cannot be either a cycle of length equal to 4. Therefore, C is a cycle of length greater than or equal to 5.

For the sake of contradiction, assume that C is not directed. This means that C contains a agent, say v, that is buying both her incident edges in C. We prove the contradiction thanks to Lemma 9, by showing that C contains a strong critical pair.

If C is an odd-length cycle, then v has two distinct antipodal vertices $u, u' \in V(C)$ which are also adjacent in C.[3] W.l.o.g., assume that u is buying the edge towards u'. Clearly, $d_G(v,u) \geq 2$. Furthermore, since C is a min cycle, it is easy to check that $\langle v,u \rangle$ is a strong critical pair.

If C is an even-length cycle, then let $u \in V(C)$ be the (unique) antipodal vertex of v and let u' be a vertex that is adjacent to u in C. Observe that $d_G(v,u), d_G(v,u') \geq 2$. Again using the fact that C is a min cycle, we have the following:

- If u is buying the edge towards u', then $\langle v,u \rangle$ is a strong critical pair.
- If u' is buying the edge towards u, then $\langle v,u' \rangle$ is a strong critical pair.

In both cases, we have proved that C contains a strong critical pair. ◀

Let (G,α) be a non-tree stable network with $n \geq 6$ vertices for a fixed $\alpha > 2n-6$ and let H be a biconnected component of G. Since $2n-6 \geq n-2$ for every $n \geq 4$, Lemma 4 implies that H cannot be a directed cycle. At the same time, if H is a cycle, then it is also a min cycle and therefore, Lemma 12 implies that H must be directed, which contradicts Lemma 4. Therefore, we have proved the following.

▶ **Corollary 13.** *For $\alpha > 2n-6$, no non-tree stable network (G,α) with $n \geq 6$ vertices contains a cycle as one of its biconnected components.*

2.4 Combining the Ingredients

Towards our main result, we start with proving that every stable network must contain a critical pair which satisfies an interesting structural property. This lemma is the third and last ingredient that is used in our analysis.

▶ **Lemma 14.** *For $\alpha > 2n-6$, every non-tree stable network (G,α) with $n \geq 6$ vertices contains a critical pair $\langle v,u \rangle$. Furthermore, there exists a path P between v and v_2 in G such that (a) the length of P is at most $2d_G(u,v)$ and (b) P uses none of the edges (v,v_1) and (v,v_2).*

Proof. Let (G,α) be a network of $n \geq 6$ vertices which is stable for a fixed $\alpha > 2n-6$, and let H be any biconnected component of G. By Corollary 13, we have that H cannot be a cycle. As a consequence, H contains at least $|V(H)|+1$ edges and, therefore, it has a vertex, say v, that buys at least two edges of H.

Let v_1 and v_2 be the two distinct vertices of H such that v buys the edges (v,v_1) and (v,v_2). Let C_i be the min cycle that contains the edge (v,v_i), whose existence is guaranteed by Lemma 11. Lemma 12 implies that C_i is a directed cycle of length greater than or equal to 5. Therefore, since (v,v_1) is an edge of C_1 bought by agent v, C_1 cannot contain the edge (v,v_2), which is also bought by v. Similarly, since (v,v_2) is an edge of C_2 bought by agent v, C_2 cannot contain the edge (v,v_1), which is also bought by v.

Let T be a shortest path tree rooted at v which gives priority to the shortest paths using the edges (v,v_1) or (v,v_2). More precisely, for every vertex x, if there is a shortest path from v to x containing the edge (v,v_1), then x is a descendant of v_1 in T. Furthermore, if no shortest path from v to x contains the edge (v,v_1), but there is a shortest path from v to x containing the edge (v,v_2), then x is a descendant of v_2 in T.

Consider the directed version of C_i in which each edge is directed from their owner agent towards the other end vertex. Let u_i be, among the vertices of C_i which are also

[3] In a cycle of length ℓ, two vertices of the cycles are *antipodal* if their distance is equal to $\lfloor \ell/2 \rfloor$.

descendants of v_i in T, the one which is in maximum distance from v w.r.t. the directed version of C_i. Finally, let (u_i, u'_i) be the edge of C_i which is bought by agent u_i. Clearly, u'_i is not a descendant of v_i in T. Therefore, by construction of T, $d_G(v, u'_i) \leq d_G(v, u_i)$, otherwise u'_i would have been a descendant of v_i in T, or there would have been a min cycle containing both edges (v, v_1) and (v, v_2) (which are both bought by agent v), thus contradicting Lemma 12.

W.l.o.g., assume that $d_G(v, u_2) \leq d_G(v, u_1)$. Let $u = u_1$ and $u' = u'_1$. We show that $\langle v, u\rangle$ is a critical pair. By Lemma 12, C_1 is a cycle of length $k \geq 5$. As C_1 is a min cycle, $k = d_G(v, u) + 1 + d_G(v, u')$. Moreover, since $d_G(v, u') \leq d_G(v, u)$, we have that $d_G(v, u) \geq \frac{k-1}{2} \geq 2$. Therefore $u' \neq v$. Next, the shortest path in T between v and u uses the edge (v, v_1) which is owned by agent v. Furthermore, the shortest path in T between v and u' does not use the edge (u, u'). Therefore, $\langle v, u\rangle$ is a critical pair.

Now, consider the path P which is obtained from C_2 by removing the edge (v, v_2). Recalling that C_2 does not contain the edge (v, v_1), it follows that P is a path between v and v_2 which uses none of the two edges (v, v_1) and (v, v_2). Therefore, recalling that $d_G(v, u'_2) \leq d_G(v, u_2)$, the overall length of P is less than or equal to

$$d_G(v, u'_2) + 1 + d_G(v_2, u_2) \leq d_G(v, u_2) + 1 + d_G(v, u_2) - 1 \leq 2d_G(v, u_1) = 2d_G(v, u). \quad \blacktriangleleft$$

Finally, we prove our main result. For this and in the rest of the paper, given a vertex x of a network (G, α) and a subset U of vertices of G, we denote by $d_G(x, U) := \sum_{x' \in U} d_G(x, x')$.

▶ **Theorem 15.** *For $\alpha > 4n - 13$, every stable network (G, α) with $n \geq 4$ vertices is a tree.*

Proof. First of all, it is easy to check that for $\alpha > 3$ every stable network with $n = 4$ vertices is a tree. Moreover, the same holds true for $n = 5$ for $\alpha > 7$.

Let $\alpha > 4n - 13$ be a fixed value and let (G, α) be a stable network with $n \geq 6$ vertices. Since $4n - 13 \geq 2n - 6$, for every $n \geq 4$, we have that if (G, α) is not a tree, then, by Lemma 14, it contains a critical pair $\langle v, u\rangle$ satisfying the conditions stated in Lemma 14. Moreover, Lemma 9 implies that $\langle v, u\rangle$ cannot be a strong critical pair. As a consequence, every shortest path from u to v_2 uses the edge (v, v_2). Since $\langle v, u\rangle$ is a critical pair, this implies that there is a shortest path from u to v_2 which uses both the edges (v_1, v) and (v, v_2). To finish our proof, we show that this contradicts the assumed stability of (G, α). This implies that (G, α) must be a tree.

Let $T(u)$ be a shortest path tree of G rooted at u having v_1 as the parent of v and v as the parent of v_2. Observe that, by definition of a critical pair, there is a shortest path between v and u containing the edge (v, v_1). Therefore, $T(u)$ is well defined. Furthermore, let X be the set of vertices which are descendants of v_2 in $T(u)$. Note that since $v_2 \in X$, we have $|X| \geq 1$.

Since $\langle v, u\rangle$ is a critical pair, thanks to Observation 6, we can use Lemma 8 (where $a = u, a' = u'$, and $b = v$) to obtain

$$\texttt{distcost}(u) \leq \texttt{distcost}(v) + n - 3. \tag{1}$$

Furthermore, observe that

$$\begin{aligned} \texttt{distcost}(u) &= \sum_{x \in X} \big(d_G(u, v) + d_G(v, x)\big) + d_G(u, V \setminus X) \\ &= d_G(u, v)|X| + d_G(v, X) + d_G(u, V \setminus X). \end{aligned} \tag{2}$$

Therefore, by substituting $\texttt{distcost}(u)$ in (1) with (2) we obtain the following

$$d_G(u, v)|X| + d_G(v, X) + d_G(u, V \setminus X) \leq \texttt{distcost}(v) + n - 3. \tag{3}$$

Let $T'(u)$ be the tree obtained from $T(u)$ by the the swap of the edge (v, v_1) with the edge (v, u). The distance cost incurred by agent v if she swaps the edge (v, v_1) with the edge (v, u) is at most

$$\begin{aligned} d_{T'(u)}(v, V) &= d_{T'(u)}(v, X) + d_{T'(u)}(v, V \setminus X) = d_{T(u)}(v, X) + d_{T'(u)}(v, V \setminus X) \\ &\leq d_{T(u)}(v, X) + \sum_{x \in V \setminus (X \cup \{v\})} \big(1 + d_{T(u)}(u, x)\big) \\ &\leq d_G(v, X) + \sum_{x \in V \setminus X} \big(1 + d_G(u, x)\big) - 2 \\ &= d_G(v, X) + n - |X| + d_G(u, V \setminus X) - 2. \end{aligned}$$

Since (G, α) is stable, agent v cannot decrease her distance cost by swapping any of the edges she owns. Therefore, we obtain

$$\texttt{distcost}(v) \leq d_G(v, X) + n - |X| + d_G(u, V \setminus X) - 2. \tag{4}$$

By summing both the left-hand and the right-hand sides of the two inequalities (3) to (4) and simplifying we obtain

$$d_G(u, v)|X| \leq 2n - 5 - |X|. \tag{5}$$

Consider the network (G', α) induced by the strategy vector in which agent v deviates from her current strategy by swapping the edge (v, v_1) with the edge (v, u) and, at the same time, by deleting the edge(v, v_2). By Lemma 14, there exists a path P between v and v_2 in G, of length at most $2d_G(u, v)$, such that P uses none of the edges (v, v_1) and (v, v_2). As a consequence, using both (1) and (5) in the second to last inequality of the following chain, the distance cost of v w.r.t. (G', α) is upper bounded by

$$\begin{aligned} d_{G'}(v, V) &\leq \sum_{x \in X} \big(2d_G(u, v) + d_G(v_2, x)\big) + \sum_{x \in V \setminus (X \cup \{v\})} \big(1 + d_G(u, x)\big) \\ &\leq 2d_G(u, v)|X| + d_G(v_2, X) + n - |X| + d_G(u, V \setminus X) - 2 \\ &\leq 2d_G(u, v)|X| + d_G(v, X) - |X| + n - |X| + d_G(u, V \setminus X) - 2 \\ &= 2d_G(u, v)|X| + d_G(u, X) - d_G(u, v)|X| + n - 2|X| + d_G(u, V \setminus X) - 2 \\ &= d_G(u, v)|X| + n - 2|X| + \texttt{distcost}(u) - 2 \\ &\leq 2n - 5 - |X| + n - 2|X| + \texttt{distcost}(v) + n - 3 - 2 \\ &= \texttt{distcost}(v) + 4n - 10 - 3|X| \leq \texttt{distcost}(v) + 4n - 13. \end{aligned}$$

By her strategy change, agent v will save α in edge cost and her distance cost will increase by at most $4n - 13$. Thus, if $\alpha > 4n - 13$, then this yields a strict cost decrease for agent v which contradicts the stability of (G, α). ◀

With the results from Fabrikant et al. [24] Theorem 15 yields:

▶ **Corollary 16.** *For $\alpha > 4n - 13$ the PoA is at most 5.*

In Section 3 we improve the upper bound of 5 on the PoA for stable tree networks from Fabrikant et al. [24]. With this, we establish the following:

▶ **Corollary 17.** *For every $\alpha > 4n - 13$ the PoA is at most $3 + \frac{2n}{2n+\alpha}$.*

3 Improved Price of Anarchy for Stable Tree Networks

In this section we show a better bound on the PoA of stable tree networks. To prove the bound, we need to introduce some new notation first. Let T be a tree on n vertices and, for a vertex v of T, let $T - v$ be the forest obtained by removing vertex v together with all its incident edges from T. We say that v is a *centroid* of T if every tree in $T - v$ has at most $n/2$ vertices. It is well known that every tree has at least one centroid vertex [28].

▶ **Lemma 18.** *Let (T, α) be a stable tree network rooted at a centroid c of T, and let $u, v \in V(T)$, with $u, v \neq c$, be two vertices such that u buys the edge towards v in T. Then $d_T(c, u) < d_T(c, v)$, i.e. u is the parent of v in T. Furthermore, if $\overline{T}$ denotes the subtree of T rooted at v, then v is a centroid of $\overline{T}$.*

We now show a useful bound on the number of vertices contained in each of the subtrees of a stable tree network rooted at a centroid.

▶ **Lemma 19.** *Let (T, α) be a stable tree network rooted at a centroid c of T, let u be a child of c in T and let v be a leaf of T contained in the subtree of T rooted at u. Let $c_1, \ldots, c_k$ be the vertices along the path in T between $c_0 = u$ and $c_k = v$, where c_{i+1} is the child of c_i, and, finally, for every $i = 1, \ldots, k$, let*

$$n_i = \left|\left\{x \in V \mid d_T(c_i, x) < d_T(c_j, x), j \neq i\right\}\right|.$$

We have that $\sum_{j=1}^{i} n_j \geq n \cdot \sum_{j=1}^{i} 1/2^j$.

We can finally prove our upper bound on the PoA of stable tree networks.

▶ **Theorem 20.** *For $\alpha \geq 2$, the PoA restricted to the class of stable tree networks of n vertices is upper bounded by $3 + \frac{2n^2 - 8n - 4\alpha}{2n^2 + (\alpha - 2)n}$.*

Proof sketch. We consider a stable tree network (T, α) rooted at a centroid vertex c. Then we consider any child c' of c and focus on a leaf vertex v in the subtree of T rooted at c'. Agent v could buy an edge to vertex c', but since (T, α) is stable, this does not yield a cost decrease for agent v. Using Lemma 19, we then compute a lower bound on the distance cost decrease if agent v buys the edge (v, c') and obtain an upper bound on the distance from any leaf v to the centroid c. This translates to an upper bound on the diameter of T. Now, using this diameter upper bound, we compute an upper bound on the social cost of (T, α) and compare it with the social cost of the socially optimal network, which is a n-node star [24], to derive the claimed bound. ◀

4 Conclusion

In this paper we have opened a new line of attack on settling the tree conjecture and on proving a constant price of anarchy for the network creation game for all α. Our technique is orthogonal to the known approaches using bounds on the average degree of vertices in a biconnected component. We are confident that our methods can be refined and/or combined with the average degree technique to obtain even better bounds – ideally proving or disproving the conjectures.

Another interesting approach is to modify our techniques to cope with the so-called max-version of the network creation game [22], where agents try to minimize their maximum distance to all other nodes, instead of minimizing the sum of distances. Also for the max-version it is still open for which α all stable networks must be trees.

References

1 Susanne Albers, Stefan Eilts, Eyal Even-Dar, Yishay Mansour, and Liam Roditty. On nash equilibria for a network creation game. *ACM TEAC*, 2(1):2, 2014.

2 Noga Alon, Erik D Demaine, Mohammad T Hajiaghayi, and Tom Leighton. Basic network creation games. *SIAM J. Disc. Math.*, 27(2):656–668, 2013.

3 Carme Àlvarez, Maria J Blesa, Amalia Duch, Arnau Messegué, and Maria Serna. Celebrity games. *Theoretical Computer Science*, 648:56–71, 2016.

4 Carme Àlvarez and Arnau Messegué. Max celebrity games. In Anthony Bonato, Fan Chung Graham, and Pawel Pralat, editors, *Algorithms and Models for the Web Graph - 13th International Workshop, WAW 2016, Montreal, QC, Canada, December 14-15, 2016, Proceedings*, volume 10088 of *Lecture Notes in Computer Science*, pages 88–99, 2016. doi:10.1007/978-3-319-49787-7_8.

5 Carme Àlvarez and Arnau Messegué. Network creation games: Structure vs anarchy. *CoRR*, abs/1706.09132, 2017. arXiv:1706.09132.

6 Nir Andelman, Michal Feldman, and Yishay Mansour. Strong price of anarchy. *Games and Economic Behavior*, 65(2):289–317, 2009. doi:10.1016/j.geb.2008.03.005.

7 Elliot Anshelevich, Onkar Bhardwaj, and Michael Usher. Friend of my friend: Network formation with two-hop benefit. *Theory Comput. Syst.*, 57(3):711–752, 2015. doi:10.1007/s00224-014-9582-4.

8 Venkatesh Bala and Sanjeev Goyal. A noncooperative model of network formation. *Econometrica*, 68(5):1181–1229, 2000.

9 Albert-László Barabási. *Network science*. Cambridge University Press, 2016.

10 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Network creation games with traceroute-based strategies. In Magnús M. Halldórsson, editor, *Structural Information and Communication Complexity - 21st International Colloquium, SIROCCO 2014, Takayama, Japan, July 23-25, 2014. Proceedings*, volume 8576 of *Lecture Notes in Computer Science*, pages 210–223. Springer, 2014. doi:10.1007/978-3-319-09620-9_17.

11 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. The max-distance network creation game on general host graphs. *Theor. Comput. Sci.*, 573:43–53, 2015. doi:10.1016/j.tcs.2015.01.044.

12 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Locality-based network creation games. *TOPC*, 3(1):6:1–6:26, 2016. doi:10.1145/2938426.

13 Davide Bilò, Luciano Gualà, and Guido Proietti. Bounded-distance network creation games. *ACM Trans. Economics and Comput.*, 3(3):16:1–16:20, 2015. doi:10.1145/2770639.

14 Davide Bilò and Pascal Lenzner. On the tree conjecture for the network creation game. *CoRR*, abs/1710.01782, 2017. arXiv:1710.01782.

15 Ulrik Brandes, Martin Hoefer, and Bobo Nick. Network creation games with disconnected equilibria. In Christos H. Papadimitriou and Shuzhong Zhang, editors, *Internet and Network Economics, 4th International Workshop, WINE 2008, Shanghai, China, December 17-20, 2008. Proceedings*, volume 5385 of *Lecture Notes in Computer Science*, pages 394–401. Springer, 2008. doi:10.1007/978-3-540-92185-1_45.

16 Ankit Chauhan, Pascal Lenzner, Anna Melnichenko, and Louise Molitor. Selfish network creation with non-uniform edge cost. In Vittorio Bilò and Michele Flammini, editors, *Algorithmic Game Theory - 10th International Symposium, SAGT 2017, L'Aquila, Italy, September 12-14, 2017, Proceedings*, volume 10504 of *Lecture Notes in Computer Science*, pages 160–172. Springer, 2017. doi:10.1007/978-3-319-66700-3_13.

17 Ankit Chauhan, Pascal Lenzner, Anna Melnichenko, and Martin Münn. On selfish creation of robust networks. In Martin Gairing and Rahul Savani, editors, *Algorithmic Game Theory - 9th International Symposium, SAGT 2016, Liverpool, UK, September 19-21, 2016.*

Proceedings, volume 9928 of *Lecture Notes in Computer Science*, pages 141–152. Springer, 2016. doi:10.1007/978-3-662-53354-3_12.

18 Jacomo Corbo and David C. Parkes. The price of selfish behavior in bilateral network formation. In Marcos Kawazoe Aguilera and James Aspnes, editors, *Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing, PODC 2005, Las Vegas, NV, USA, July 17-20, 2005*, pages 99–107. ACM, 2005. doi:10.1145/1073814.1073833.

19 Andreas Cord-Landwehr and Pascal Lenzner. Network creation games: Think global - act local. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, volume 9235 of *Lecture Notes in Computer Science*, pages 248–260. Springer, 2015. doi:10.1007/978-3-662-48054-0_21.

20 Andreas Cord-Landwehr, Alexander Mäcker, and Friedhelm Meyer auf der Heide. Quality of service in network creation games. In Tie-Yan Liu, Qi Qi, and Yinyu Ye, editors, *Web and Internet Economics - 10th International Conference, WINE 2014, Beijing, China, December 14-17, 2014. Proceedings*, volume 8877 of *Lecture Notes in Computer Science*, pages 423–428. Springer, 2014. doi:10.1007/978-3-319-13129-0_34.

21 Erik D. Demaine, Mohammad Taghi Hajiaghayi, Hamid Mahini, and Morteza Zadimoghaddam. The price of anarchy in cooperative network creation games. *SIGecom Exch.*, 8(2):2:1–2:20, 2009.

22 Erik D. Demaine, Mohammad Taghi Hajiaghayi, Hamid Mahini, and Morteza Zadimoghaddam. The price of anarchy in network creation games. *ACM Trans. Algorithms*, 8(2):13:1–13:13, 2012. doi:10.1145/2151171.2151176.

23 Shayan Ehsani, Saber Shokat Fadaee, MohammadAmin Fazli, Abbas Mehrabian, Sina Sadeghian Sadeghabad, Mohammad Ali Safari, and Morteza Saghafian. A bounded budget network creation game. *ACM Trans. Algorithms*, 11(4):34:1–34:25, 2015. doi:10.1145/2701615.

24 Alex Fabrikant, Ankur Luthra, Elitza N. Maneva, Christos H. Papadimitriou, and Scott Shenker. On a network creation game. In Elizabeth Borowsky and Sergio Rajsbaum, editors, *Proceedings of the Twenty-Second ACM Symposium on Principles of Distributed Computing, PODC 2003, Boston, Massachusetts, USA, July 13-16, 2003*, pages 347–351. ACM, 2003. doi:10.1145/872035.872088.

25 Sanjeev Goyal, Shahin Jabbari, Michael J. Kearns, Sanjeev Khanna, and Jamie Morgenstern. Strategic network formation with attack and immunization. In Yang Cai and Adrian Vetta, editors, *Web and Internet Economics - 12th International Conference, WINE 2016, Montreal, Canada, December 11-14, 2016, Proceedings*, volume 10123 of *Lecture Notes in Computer Science*, pages 429–443. Springer, 2016. doi:10.1007/978-3-662-54110-4_30.

26 Matthew O Jackson and Asher Wolinsky. A strategic model of social and economic networks. *Journal of economic theory*, 71(1):44–74, 1996.

27 Tomasz Janus and Bart de Keijzer. On strong equilibria and improvement dynamics in network creation games. In Nikhil R. Devanur and Pinyan Lu, editors, *Web and Internet Economics - 13th International Conference, WINE 2017, Bangalore, India, December 17-20, 2017, Proceedings*, volume 10660 of *Lecture Notes in Computer Science*, pages 161–176. Springer, 2017. doi:10.1007/978-3-319-71924-5_12.

28 O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. ii: The p-medians. *SIAM Journal on Applied Mathematics*, 37(3):pp. 539–560, 1979.

29 Bernd Kawald and Pascal Lenzner. On dynamics in selfish network creation. In Guy E. Blelloch and Berthold Vöcking, editors, *25th ACM Symposium on Parallelism in Algorithms*

and Architectures, SPAA '13, Montreal, QC, Canada - July 23 - 25, 2013, pages 83–92. ACM, 2013. doi:10.1145/2486159.2486185.

30 Jon M. Kleinberg. The small-world phenomenon: an algorithmic perspective. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 163–170. ACM, 2000. doi:10.1145/335305.335325.

31 Lasse Kliemann. The price of anarchy for network formation in an adversary model. *Games*, 2(3):302–332, 2011. doi:10.3390/g2030302.

32 Elias Koutsoupias and Christos H. Papadimitriou. Worst-case equilibria. In Christoph Meinel and Sophie Tison, editors, *STACS 99, 16th Annual Symposium on Theoretical Aspects of Computer Science, Trier, Germany, March 4-6, 1999, Proceedings*, volume 1563 of *Lecture Notes in Computer Science*, pages 404–413. Springer, 1999. doi:10.1007/3-540-49116-3_38.

33 Pascal Lenzner. On dynamics in basic network creation games. In Giuseppe Persiano, editor, *Algorithmic Game Theory, 4th International Symposium, SAGT 2011, Amalfi, Italy, October 17-19, 2011. Proceedings*, volume 6982 of *Lecture Notes in Computer Science*, pages 254–265. Springer, 2011. doi:10.1007/978-3-642-24829-0_23.

34 Pascal Lenzner. Greedy selfish network creation. In Paul W. Goldberg, editor, *Internet and Network Economics - 8th International Workshop, WINE 2012, Liverpool, UK, December 10-12, 2012. Proceedings*, volume 7695 of *Lecture Notes in Computer Science*, pages 142–155. Springer, 2012. doi:10.1007/978-3-642-35311-6_11.

35 Pascal Lenzner. *On selfish network creation.* PhD thesis, Humboldt University of Berlin, 2014. URL: http://edoc.hu-berlin.de/dissertationen/lenzner-pascal-2014-05-30/PDF/lenzner.pdf.

36 Akaki Mamageishvili, Matús Mihalák, and Dominik Müller. Tree nash equilibria in the network creation game. *Internet Mathematics*, 11(4-5):472–486, 2015. doi:10.1080/15427951.2015.1016248.

37 Eli A. Meirom, Shie Mannor, and Ariel Orda. Network formation games with heterogeneous players and the internet structure. In Moshe Babaioff, Vincent Conitzer, and David Easley, editors, *ACM Conference on Economics and Computation, EC '14, Stanford , CA, USA, June 8-12, 2014*, pages 735–752. ACM, 2014. doi:10.1145/2600057.2602862.

38 Matús Mihalák and Jan Christoph Schlegel. Asymmetric swap-equilibrium: A unifying equilibrium concept for network creation games. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of *Lecture Notes in Computer Science*, pages 693–704. Springer, 2012. doi:10.1007/978-3-642-32589-2_60.

39 Matús Mihalák and Jan Christoph Schlegel. The price of anarchy in network creation games is (mostly) constant. *Theory Comput. Syst.*, 53(1):53–72, 2013. doi:10.1007/s00224-013-9459-y.

40 Jeffrey Travers and Stanley Milgram. The small world problem. *Phychology Today*, 1:61–67, 1967.

On Low for Speed Oracles

Laurent Bienvenu
LIRMM, CNRS & Université de Montpellier, Montpellier, France
laurent.bienvenu@computability.fr

Rodney Downey
School of Mathematics and Statistics, Victoria University of Wellington, Wellington, New Zealand
rod.downey@vuw.ac.nz

Abstract

Relativizing computations of Turing machines to an oracle is a central concept in the theory of computation, both in complexity theory and in computability theory(!). Inspired by lowness notions from computability theory, Allender introduced the concept of "low for speed" oracles. An oracle A is low for speed if relativizing to A has essentially no effect on computational complexity, meaning that if a decidable language can be decided in time $f(n)$ with access to oracle A, then it can be decided in time $poly(f(n))$ without any oracle. The existence of non-computable such A's was later proven by Bayer and Slaman, who even constructed a computably enumerable one, and exhibited a number of properties of these oracles as well as interesting connections with computability theory. In this paper, we pursue this line of research, answering the questions left by Bayer and Slaman and give further evidence that the structure of the class of low for speed oracles is a very rich one.

2012 ACM Subject Classification Theory of computation → Turing machines, Theory of computation → Complexity classes

Keywords and phrases Lowness for Speed, Oracle Computations, Turing Degrees

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.15

Related Version A full version of the paper is available at `https://arxiv.org/abs/1712.09710`.

Funding Bienvenu acknowledges support of ANR-15- CE40-0016-01 RaCAF grant, Downey thanks the Marsden Fund of New Zealand, and the LIRMM (University of Montpellier) where this research was undertaken.

Acknowledgements We would like to thank three anonymous referees for their helpful comments and suggestions.

1 Introduction

The subject of this paper is oracle computation, more specifically the effect of oracles on the speed of computation. There are many notable results about oracles in classical complexity, beginning with the Baker-Gill-Solovay result [3] which asserts that there are oracles A such that $\mathsf{P}^A = \mathsf{NP}^A$, but that there are also oracles B such that $\mathsf{P}^B \neq \mathsf{NP}^B$ (thus demonstrating that methods that relativize will not suffice to solve basic questions like P vs NP). An underlying question is whether oracle results can say things about complexity questions in the unrelativized world. Eric Allender and his co-authors [1, 2] showed that oracle access to the sets of random strings could give insight into basic complexity questions. For example,

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 15; pp. 15:1–15:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


in [2], Allender et. al. showed that $\cap_U \mathsf{P}^{R_{K_U}} \cap \mathsf{COMP} \subseteq \mathsf{PSPACE}$ where R_{K_U} denotes the strings whose prefix-free Kolmogorov complexity (relative to universal machine U) is at least their length, and COMP denotes the collection of computable sets. Later the "$\cap\mathsf{COMP}$" was removed by Cai et. al. [7]. Thus we conclude that reductions to very complex sets like the random strings somehow gives insight into very simple things like computable sets.

Inspired by lowness notions in computability theory, Allender asked whether there were non-trivial sets which were "low for speed" in that, as oracles, they did not accelerate running times of computations by more than a polynomial amount. Of course, as stated this makes little sense since using any X as oracle, we can decide membership in X in linear time, while without oracle X may not even be computable at all! Thus, what we are really interested in is the set of oracles which do not speed-up the computation of *computable sets* by more than a polynomial amount. More precisely, an oracle X is *low for speed* if for any computable language L, if some Turing machine M with access to oracle X decides L in time f, then there is a Turing machine M' without oracle and polynomial p such that M' decides L in time $p \circ f$. (Here computation time of oracle computation is counted in the usual complexity-theoretic fashion: we have a "query tape" on which we can write strings, and once a string x is written on this tape, we get to ask the oracle whether x belongs to it in time $O(1)$).

There are trivial examples of such sets, namely oracles that belong to P, because any query to such an oracle can be replaced by a polynomial-time computation. Allender's precise question was therefore:

Is there an oracle $X \notin \mathsf{P}$ which is low for speed?

Such an X, if it exists, has to be non-computable, for the same reason as above (if X is computable and low for speed, then X is decidable in linear time using oracle X, thus – by lowness – decidable in polynomial time without oracle, i.e., $X \in \mathsf{P}$).

A partial answer was given by Lance Fortnow (unpublished), who observed the following.

▶ **Theorem 1** (Fortnow). *If X is a hypersimple and computably enumerable oracle, then X is low for polynomial time, in that if $L \in \mathsf{P}^X$, then $L \in \mathsf{P}$.*

Allender's question was finally solved by Bayer and Slaman, who showed the following.

▶ **Theorem 2** (Bayer-Slaman [4]). *There are non-computable, computably enumerable, sets X which are low for speed.*

Once their existence is established, it is natural to wonder what kind of sets might be low for speed. A precise characterization seems currently out of reach, but it is interesting to see how lowness for speed interacts with other computability-theoretic properties. One needs however to keep in mind that lowness for speed is *not* closed under Turing equivalence: we saw above that in the $\mathbf{0}$ degree (computable sets) some members are low for speed and others that are not (on the other hand it is easy to see that if A is polynomial-time reducible to B and B is low for speed, then A is also low for speed).

In his PhD thesis, Bayer showed that if X is computably enumerable and of promptly simple Turing degree, then X is *not* low for speed, but also proved that this did not characterize the computable enumerable oracles that are low for speed. Bayer also studied the size of the set of low for speed oracles, where 'size' is understood in terms of Baire category. Surprisingly, whether the set of low for speed oracles is meager or co-meager depends on the answer of the famous $\mathsf{P} =? \mathsf{NP}$ question.

In this paper, we continue Bayer and Slaman's investigation on the set of low for speed oracles. In the next section, we give an easier proof of the existence of non-computable low

for speed oracles which does not require the full Bayer-Slaman machinery (but the oracle we construct is not computably enumerable). In Section 3, we focus on the computably enumerable low for speed oracles, and prove that – quite surprisingly – they cannot be low in the computability-theoretic sense, but can however be low_2. Finally, we pursue Bayer and Slaman's idea to study how large the set of low for speed oracles is, in terms of measure and category. In particular, we solve a question they left open by showing that the set of low for speed oracles has measure 0 and obtain some interesting connections with algorithmic randomness. Finally, though lowness for speed is not closed under Turing equivalence, it is nonetheless natural to ask which Turing degrees contain a low for speed member, which is what Section 5 is about.

Throughout this paper, we will denote by $\{0,1\}^*$ the set of finite strings. In our setting, an oracle is a *language*, i.e., a subset of $\{0,1\}^*$; however, as is typical in computability theory, it is more convenient in some of the results we present below to view oracles as infinite binary sequences (whose set we denote by $\{0,1\}^\omega$), by first identifying finite strings with integers (the $(n+1)$-th string in the length-lexicographic order being identified with n) making the oracle a subset of $\mathbb{N}$ and then identifying the oracle with its characteristic sequence (the $(n+1)$-th bit is 1 if n belongs to the oracle, 0 otherwise). When building oracles X with certain computability-theoretic properties, viewed as infinite binary sequences, we will often need to refer to prefixes of X, which are themselves binary strings. To avoid confusion between *members* and *prefixes* of oracles, we will use latin letters $x, y, z, \dots$ to denote members of oracles, and greek letters $\sigma, \tau, \dots$ for prefixes of oracles. Two strings σ and τ are *incompatible* if for some $i < \min(|\sigma|, |\tau|)$, $\sigma(i) \neq \tau(i)$. We denote this by $\sigma \perp \tau$. The join $X \oplus Y$ of two infinite binary sequences X, Y is the sequence $X(0)Y(0)X(1)Y(1)\dots$. Finally $X \upharpoonright n$ is the prefix of X of length n.

Our paper requires some knowledge of computability theory and algorithmic randomness. One can consult the book [8] for the results and concepts we allude to below. Our notation is mostly standard. We fix a computable bijection $\langle ., . \rangle$ from pairs of strings/integers to strings/integers. We also fix an effective list (Φ_e) of all oracle Turing functionals (or machines: Φ_e^A is the Turing machine of index e with oracle A, which for a fixed A is a partial function from $\{0,1\}^*$ to $\{0,1\}$). For a given functional Φ_e and oracle A, $time(\Phi_e^A, x)$ denotes the running time of Φ_e on input x with oracle A (counting time according to the model of computation described above) and $time(\Phi_e^A)$ is the function $x \mapsto time(\Phi_e^A, x)$. We let (R_i) be an effective enumeration of all partial computable functions from $\{0,1\}^*$ to $\{0,1\}$. We denote the set of low for speed oracles by LFS, and the subset of LFS consisting of its non-computable elements by LFS^*.

Due to space restrictions some proofs are omitted. They can be found in the extended version of the paper, available at `https://arxiv.org/abs/1712.09710`.

2 Existence of non-computable low for speed sets

In this section we will present a simple proof of the existence of a non-computable low for speed oracle. Define the set S of strings by $S = \{0^{2^n} \mid n \in \mathbb{N}\}$ and – identifying S with a set of integers as discussed above – let $\mathbb{S}$ be the set of 'sparse' infinite binary sequences (viewed as sets of integers) that only contain elements from S, that is, $\mathbb{S} = \{X \in \{0,1\}^\omega \mid X \subseteq S\}$.

By extension, we say that a string σ is in $\mathbb{S}$ if it is a prefix of some element of $\mathbb{S}$. The interest of the set $\mathbb{S}$ is that there are only $O(n)$ strings in $\mathbb{S}$ of length n. Thus, given a Turing machine Φ, it is possible to simulate in time $poly(t)$ the behaviour of Φ^X during t steps of computation on all $X \in \mathbb{S}$ (an idea that is already present in the Bayer-Slaman argument presented in the next section).

▶ **Theorem 3.** *There exists a non-computable X which is low for speed.*

Proof. We want X to satisfy all requirements $\mathcal{R}_{(e,i)}$, defined as follows:

$\mathcal{R}_{(e,i)}$: either R_i is partial, or $\Phi_e^X \neq R_i$, or $\Phi_e^X = R_i$ but the computation of R_i via Φ_e^X can be simulated by a functional Ψ running in time polynomial in $time(\Phi_e^X)$.

We build our oracle X by finite extension. Let σ_0 be the empty string. At stage $s+1 = \langle e, i\rangle$, do the following.

(a) If there is an n and a $\tau \in \mathbb{S}$ extending σ_s such that $\Phi_e^\tau(n)$ and $R_i(n)$ both converge and have different values, then let σ_{s+1} be the first (say in length-lexicographic order) such string τ.

(b) If there is no such string τ, then take $\sigma_{s+1} = \sigma_s 0$

Finally let X be the unique infinite sequence extending all σ_s. We claim that X is as wanted. Let us first prove that X must be incomputable. Suppose $X = R_i$ for a total R_i. Let e be an index such that Φ_e is the identity functional. By construction, when choosing the prefix τ of X at stage $s+1 = \langle e, i\rangle$, we must be in case (a), and thus τ is precisely chosen to ensure $X \neq R_i$, a contradiction. Let us now prove that X is low for speed. Fix a pair (e, i) let $s+1 = \langle e, i\rangle$, and let us see how σ_{s+1} was constructed. If we were in case (a) at that stage, we have ensured $\Phi_e^{\sigma_{s+1}} \perp R_i$ and thus $\Phi_e^X \perp R_i$, thereby satisfying $\mathcal{R}_{(e,i)}$. If we were in case (b), there are three subcases:

- Either R_i is partial, then the requirement $\mathcal{R}_{(e,i)}$ is satisfied.
- Or there is an n such that $\Phi_e^\tau(n) \uparrow$ for any extension τ of σ_s, in which case $\Phi_e^X(n) \uparrow$ and thus $\Phi_e^X \neq R_i$ should R_i be total.
- Or, if we are in neither of the two above cases, for every n there is an extension τ of σ_s such that $\Phi_e^\tau(n) \downarrow$, and for any such τ, we have $\Phi_e^\tau(n) = R_i(n)$. In this case, we can build a functional Ψ which computes R_i as follows. On input n, at stage t, it computes $\Phi_e^\tau(n)$ during t steps of computation for all $\tau \in \mathbb{S}$ of length t extending σ_s. If a τ is found such that $\Phi_e^\tau(n) \downarrow$, then we set $\Psi(n) = \Phi_e^\tau(n)$. As we already mentioned, there are only $O(t)$ strings of length t in $\mathbb{S}$ and it is obvious that they can be listed in polynomial time. Hence, simulating all computations $\Phi_e^\tau(n)$ during t steps can be done in time $p(t)$ for some polynomial t. This shows that for any $Y \in \mathbb{S}$ extending σ_s, if $\Phi_e^Y(n)$ returns (the value of $R_i(n)$) in time t, this is found out by the procedure Ψ at stage t, which corresponds to $\sum_{s \leq t} p(s) + O(1)$ steps of computation for Ψ, which is also polynomial in t. This being true for any $Y \in \mathbb{S}$ extending σ_s, we have in particular that $time(\Psi) = poly(time(\Phi_e^X))$. ◀

One should note that the case disjunction in this proof is a Σ_1/Π_1 dichotomy, and therefore one can carry out the construction below $\mathbf{0}'$, therefore establishing the existence of a $\mathbf{0}'$-computable set that is low for speed. This is weaker than the Bayer-Slaman result presented in the next section, which asserts the existence of a *c.e.* such set. However, this proof is both simpler and, as we will see in the remainder of the paper, has further useful corollaries.

3 Computably enumerable low for speed sets

We now restrict ourselves to the computably enumerable (c.e.) sets, and study which of these can be low for speed. For the sake of completeness, we present the main ideas of the proof of Bayer and Slaman [4] that there are indeed c.e. sets in LFS^*.

▶ **Theorem 4** (Bayer-Slaman Theorem). *There exist c.e. non-computable sets that are low for speed.*

Proof sketch. The proof uses a tree-of-strategies argument. We need to satisfy

$$\mathcal{P}_e : \overline{A} \neq W_e,$$

and

$$\mathcal{L}_{e,i} : \text{If } \Phi_e^A = R_i \text{ total, then some } \Psi \text{ computes } R_i \text{ in time polynomial in } time(\Phi_e^A).$$

The $\mathcal{P}_e$-strategy is a standard Friedberg-Muchnik strategy on a tree. A node ρ devoted to this requirement picks a fresh follower x, waits for $x \in W_e[s]$ and if this happens puts x into A.

The basic strategy for $\mathcal{L}_{e,i}$ is the following. First, throughout the whole construction of A, we will promise that if we add an element x to A at stage t, then we must immediately also add all $y \in [x, t]$ (this is often referred to as a *dump construction*). This way, at any stage s, there will only be at most s strings α of length s that can potentially be a prefix of (the final) A. And thus – just like in the previous section – at stage s, it is possible to emulate all computations $\Phi_e^\alpha(x)[s]$ for all such α's and $x \leq s$ in time $poly(s)$.

When the strategy is eligible to act at stage s, for every $x \leq s$ on which Ψ is not defined yet, it computes all $\Phi_e^\alpha(x)[s]$ for all potential prefixes α of A, and should one of them converge, defines $\Psi(x)$ to be the value of $\Phi_e^\alpha(x)$ for the α that has the fastest convergence. If no $\Phi_e^\alpha(x)[s]$ converges, $\Psi(x)$ remains undefined until the strategy is eligible to act again.

Now, if at some later stage we find a value x such that $R_i(x) \downarrow$ and $\Psi(x) \neq R_i(x)$, then we find the α such that $\Psi(x) = \Phi_e^\alpha$ and add elements into A so that α becomes a prefix of A. This ensures $\Phi_e^A \neq R_i$ and terminates the strategy. All strategies of lower priorities are then injured and must be reset. If we never find such an x, this means that either Φ_e^A is partial, or R_i is, or $\Psi = \Phi_e^A = R_i$ and by construction the running time of Ψ is polynomial in the running time of Φ_e^A.[1]

This is enough to ensure the success of the strategy in isolation. The difficulty comes from the interaction with lower-priority strategies which might want to add elements into A. The final key to the Bayer-Slaman proof is the following. Suppose that at some stage s a strategy of lower priority wants to add an interval $[y, s]$ of elements into A. The problem is that the computations on this configuration might be *slow*. Perhaps for some x of length $\leq s$ we have not as yet seen $\Phi_e^{A_s \cup [y,s]}(x) \downarrow$. Even more importantly, we don't even know that the value of this will agree with the value $\Psi(x)$ we have already defined.

The idea is the following. R_i has to confirm the computations, that is, we wait until $R_i(x)$ converges on all x where Ψ has already been defined. When (and if) this happens, we must have $\Psi(x) = R_i(x)$ for all such x otherwise we would be in the above case where we can ensure $\Phi_e^A \neq R_i$ and satisfy the requirement. If this never happens, our requirement will be satisfied because R_i would be partial. And if it does happen, then we can safely add $[y, s]$ to A because if this causes $\Phi_e^A(x)$ to change, it will yield $\Phi_e^A(x) \neq R_i(x)$ which satisfies the requirement. But there is one last problem: while waiting for this confirmation, the construction of Ψ cannot wait as we need it to be as fast as Φ_e^A. The crucial trick is, from the point of view of our strategy, to carry on *as if* $[y, s]$ had already been enumerated into A.

[1] Actually, there is a subtlety here: one must ensure that the strategy for $\mathcal{L}_{e,i}$ is eligible to act often enough, i.e., allowed to act for the n-th time before stage $q(n)$ for some polynomial q, but this can easily be ensured.

Indeed, if the confirmation ever happens, the elements of $[y, s]$ will be truly enumerated into A which Ψ will have correctly assumed ahead of time, and if the confirmation never happens, Ψ might be wrong (i.e., $\Psi \neq \Phi_e^A$) but this will not matter because in this case R_i will be either partial or different from Φ_e^A. Of course, in the case where the confirmation never occurs the strategy of lower priority never gets to enumerate into A the elements it wants. This is where we make use of a standard tree construction where strategies of lower priorities guess the outcome of strategies of higher priority. We refer the reader to [4] for details. ◀

Within the c.e. sets, one would expect that a low for speed c.e. set would be one with little computational power, in the same way that sets low for 1-randomness are all (super-)low (see Nies [14]). The next theorem is therefore quite surprising.

▶ **Theorem 5.** *If A is non-computable, c.e., and of low Turing degree (i.e. $A' \equiv_T \emptyset'$), then A is* not *low for speed.*

Proof. Assume that A is not computable, is c.e., and is low. Let (Φ_e, p_e) be an enumeration of pairs of one functional and one polynomial with coefficients in $\mathbb{N}$. We will build a Turing functional Ψ and a computable set R such that $\Psi^A = R$. This is our global requirement and we make the following global commitment: if a value $R(n)$ gets defined at some stage, $\Psi^X(n)$ is immediately defined to be equal to $R(n)$ for all X's on which $\Psi^X(n)$ is still undefined. We want to satisfy, for each e:

$(\mathcal{R}_e)$: Φ_e does not compute R in time $p_e(time(\Psi^A))$

thus proving that A is not low for speed. We give a strategy for a single requirement $(\mathcal{R}_e)$ (the strategies for different requirements interact to the extent that each one needs to know the actions of the others in order to pick fresh witnesses, but the construction is injury-free). Throughout the construction, we build a 'verifier', i.e., a partial computable S such that $S(e, .)$ is the attempt by the $(\mathcal{R}_e)$-strategy to guess A. We also define an auxiliary functional Θ common to all strategies whose index we know in advance, and use the lowness of A to obtain a computable 0-1 valued function $h(., .)$ such that $\lim_t h(e, t)$ exists for all e, and equals 1 when $\Theta^A(e) \downarrow$, 0 otherwise. (Informally, $\Theta^X(e) \downarrow$ means that a prefix of X is believed to be a prefix of A at some stage of the strategy for $(\mathcal{R}_e)$, and this will cause the strategy to enter Case 3 as described below.)

At the initial stage s_1, S is empty and we pick a first fresh witness w_1 larger than any integer mentioned so far in the construction and define $\Psi^{A_{s_1}\restriction 1}(w_1) = 0$. Let t_1 be the time this computation takes. Now, check whether $\Phi_e(w_1)$ returns in $p_e(t_1)$ steps. We distinguish three cases:

Case 1: $\Phi_e(w_1)$ returns 1 in $\leq p_e(t_1)$ steps. In this case, we set $R(w_1) = 0$ and $R(n) = 0$ for all $n \leq w_1$ on which R is still undefined, and commit to having $\Psi^A(w_1) = 0$ even after potential future A-changes. This way we ensure $\Phi_e \neq R = \Psi^A$, thus immediately satisfying $(\mathcal{R}_e)$, and we stop the strategy for this requirement.

Case 2: $\Phi_e(w_1)$ returns 0 in $\leq p_e(t_1)$ steps. In this case, we do not define $R(w_1)$ just yet. Instead, we set $S(e, 1) = A_{s_1} \restriction 1$. We then create a second witness w_2 at stage s_2 and proceed as above for this new witness (with $A_{s_2} \restriction 2$ in place of $A_{s_1} \restriction 1$). And so on: for further occurrences of this case, the procedure will extend S and create a witness w_3 at stage s_3 looking at prefixes of length $l = 3$, etc (and if Case 2 then causes a reset, we stay at the same level l, that is, keep the same l, when resetting). Meanwhile, we continue to monitor A. Again, there are two subcases for a given l:

(a) At some point we discover that $A_{s_l} \upharpoonright l$ is not in fact an initial segment of A, we are then free to set $R(w_l) = 1$ (which will guarantee $\Psi^A(w_l) = 1 = R(w_l) \neq \Phi_e(w_l)$ since we only committed to $\Psi^\sigma(w_l) = 0$ for σ's that are not prefixes of A), and this way we have satisfied $(\mathcal{R}_e)$. We then stop the strategy.

(b) $A_{s_l} \upharpoonright l$ is a true initial segment of A, in which case nothing further will happen regarding witness w_l. What is gained is that $S(e, l)$ will be defined to be $A_{s_l} \upharpoonright l = A \upharpoonright l$, thus progress was made towards computing A.

Case 3: $\Phi_e(w_1)$ is still undefined after $p_e(t_1)$ steps. In this case, we set $\Theta^{A_{s_1}\upharpoonright 1}(e) \downarrow$ (which should be interpreted as signalling that we are currently in Case 3). Observe that if $A_{s_1} \upharpoonright 1$ is a true prefix of A, this implies $\Theta^A(e) \downarrow$ and therefore we would have $\lim h(e, t) = 1$. We distinguish two subcases.

(a) The current value $h(e, s)$ is 0. Then we wait for a stage $t > s$ such that either $h(e, t) = 1$ or $A_t \upharpoonright 1 \neq A_s \upharpoonright 1$ (one of the two must happen as we explained above). If the former happens first we move to subcase (b) below. If the latter happens first, we restart the procedure, resetting s_1 to the current stage t and keeping the same w_1.

(b) The current value $h(e, s)$ is 1. We then set $R(w_1) = 0$, set $R(n) = 0$ for all $n \leq w_1$ on which R is still undefined and terminate the strategy *for now*. However, if at a later time $t > s$, we see that $h(e, t) = 0$ *and* $A_t \upharpoonright 1 \neq A_{s_1} \upharpoonright 1$, then we resurrect the strategy and start over at the level l where we left off.

We claim that this strategy satisfies the requirement $(\mathcal{R}_e)$. If Case 1 happens for any witness w_l, the requirement is satisfied. Case 3a can only happen finitely many times at a given level since as $A_{s_l} \upharpoonright l$ can only change finitely many times. Case 3b can only happen finitely many times across all levels as each passage through this case causes a flip of $h(e, .)$, and we know $h(e, .)$ converges. Case 2b can also happen only finitely often, because each time we go through this case and do not get to diagonalize, $S(e, .)$ computes a longer initial segment of A, but A is incomputable so $S(e, l)$ would eventually have to be wrong.

Thus we either eventually end up in Case 1 (and immediately succeed) or Case 2a (and immediately succeed) or a terminal Case 3b, i.e., the strategy enters Case 3b and stays there forever. It remains to check this last scenario. Suppose the terminal Case 3b happens for some $A_{s_l} \upharpoonright l$ which is not a prefix of A, this means that $\Psi^A(w_l)$ has not been defined yet and thus, should nothing else happen, we would have $\lim_t h(e, t) = 0$ and would see a change in $A \upharpoonright l$, thus leaving this occurrence of Case 3b, a contradiction. So $A_{s_l} \upharpoonright l$ is indeed a prefix of A and by construction $\Psi^A(w_l)$ returns $0 = R(i)$ in a number of steps t while $\Phi_e(w_l)$ does not return in less than $p_e(t)$ steps, thus the requirement is satisfied. It is now straightforward to satisfy all requirements by ordering them in order of priority, noticing that each strategy only makes finitely many changes to R before achieving its goal and R is total as every time $R(w)$ becomes defined, so do the $R(n)$ for $n \leq w$ that were previously undefined. ◀

It is important to note that Theorem 5 fails to hold outside of the c.e. setting.

▶ **Theorem 6.** *There exists a low, non-computable set X which is low for speed.*

Proof. See Section 5. ◀

One can also show that Theorem 5 does not extend to other levels of the 'low' hierarchy within c.e. sets.

▶ **Theorem 7.** *There is a low$_2$ c.e. set that is low for speed.*

We can also combine the the same ideas (dump construction together with awaiting for certification) with the standard proof that there exists an incomplete c.e. set A of high Turing degree (i.e., $A' \equiv_T \emptyset''$) to get the following.

▶ **Theorem 8.** *There is a high c.e. set A which is low for speed.*

4 How big is LFS?

Bayer and Slaman showed that whether LFS is meager or not... depends on the answer to P vs NP question! More precisely, if P = NP, then LFS is co-meager (even more precisely, every 2-generic is low for speed, see [8, Section 2.24] for a discussion of the various notions of genericity), while if P ≠ NP, then LFS is meager (more precisely, every recursively generic is not low for speed). They left as an open question whether LFS has measure (Lebesgue) 0 or 1 (by Kolmogorov's 0/1-law, it has to be one or the other). One might expect that, just like the meagerness of LFS depends on the P vs NP question, its measure depends on complexity-theoretic assumptions, such as the 'P vs BPP' question. This is not the case: we show that LFS is – unconditionally – a nullset.

▶ **Theorem 9.** *The set* LFS *has measure* 0.

We postpone the proof of this theorem to the next section as it builds upon the proof of Theorem 15.

On the other hand, we will see in the next section that LFS is large in the set-theoretic sense, namely that it has the size of the continuum.

Finally, there is one last notion of size for subsets of $\{0,1\}^\omega$ that is dear to computability theorists, namely, a set is 'large' if it contains a Turing upper cone and is 'small' if it disjoint from a Turing upper cone. Martin's Turing determinacy theorem tells us that any Borel set which is closed under Turing equivalence must be either large or small on this account. The set LFS is indeed Borel (this is easy to see from the definition), but it is not closed under Turing equivalence, so Martin's theorem does not apply. In the next section, we will use a classical result from complexity theory to show that LFS is in fact disjoint from a Turing upper cone (Theorem 15).

5 Lowness for speed and Turing degrees

While lowness for speed is not closed under Turing equivalence, the following question is nonetheless interesting:

Which sets are Turing equivalent to some low for speed X? Which sets compute some non-computable low for speed X?

We denote by **LFS** and **LFS*** the set of Turing degrees that contain a low for speed set and a non-computable low for speed set, respectively. One of the main results of Bayer [4] is that not all degrees are in **LFS**. Indeed, there exists a c.e. degree $\mathbf{a} \notin \mathbf{LFS}$. The main question left open by Bayer regarding **LFS** is whether it is downward closed under $\leq_T$ or closed under join. We give a negative answer to both questions. To show that it is not downward closed, we need the following extension of Theorem 5 to degrees.

▶ **Theorem 10.** *For any low c.e. degree $\mathbf{a} > \mathbf{0}$, we have $\mathbf{a} \notin \mathbf{LFS}$.*

▶ **Corollary 11.** **LFS** *is not downward closed under* $\leq_T$*, even within c.e. degrees.*

Proof. Let $\mathbf{a} > \mathbf{0}$ be a c.e. degree in **LFS** whose existence was explained in Section 3. By Sacks's splitting theorem [15], there is a low c.e. degree $\mathbf{0} < \mathbf{b} < \mathbf{a}$. By Theorem 10, $\mathbf{b} \notin \mathbf{LFS}$. ◀

The next theorem will show that while not every c.e. degree contains a low for speed member, every non-zero c.e. degree $\mathbf{a}$ *bounds* a degree $\mathbf{b} \in \mathbf{LFS}$. Recall Bayer's result that whether 2-generics are low for speed or not depends on the 'P vs NP' question. When it comes to the *degree* of generics, we have that every 1-generic is Turing-equivalent to a set that is low for speed, independently of complexity-theoretic assumptions.

▶ **Theorem 12.** *Every* 1*-generic degree* $\mathbf{g}$ *belongs to* $\mathbf{LFS}^*$*.*

Proof. We get this result by refining the proof of Theorem 3. In that proof, we built an X low for speed by finite extension, and ensuring that X was a subset of $S = \{0^{2^n} \mid n \in \mathbb{N}\}$. For $G \subseteq \mathbb{N}$, let $S_G = \{0^{2^n} \mid n \in G\}$. We claim that when G is 1-generic, $S_G = \{0^{2^n} \mid n \in G\}$ is low for speed (and clearly $S_G \equiv_T G$). In the proof of Theorem 3, if we let $\mathcal{U}_{e,i}$ be the effectively open set of those Z such that for some n, $\Phi_e^{S_Z}(n)$ and $R_i(n)$ both converge to different values, we know that G, being 1-generic, is either in $\mathcal{U}_{e,i}$ (hence satisfying the requirement $\mathcal{R}_{e,i}$ as per case (a)), or in the interior of the complement of $\mathcal{U}_{e,i}$, which precisely corresponds to case (b), hence the requirement is also satisfied in this case. ◀

We can derive a number of useful corollaries from this theorem. First of all, we see that LFS has the size of the continuum since $G \mapsto S_G$ is one-to-one, and there are continuum many 1-generic G. We also get an immediate proof of Theorem 6 that asserts the existence of a set of low degree that is low for speed.

Proof of Theorem 6. Take a Δ^0_2 1-generic G; the corresponding set S_G is low for speed and is low because it is both Δ^0_2 and GL_1 (Indeed a result of Jockusch [10] states that every 1-generic degree G is GL_1, that is, $G' \equiv_T G \oplus \emptyset'$; when $G \leq_T \emptyset'$, this is equivalent to $G' \equiv_T \emptyset'$). ◀

A similar idea allows us to show that **LFS** contains a non-trivial interval in the Turing degrees.

▶ **Corollary 13.** *There is a degree* $\mathbf{a} > \mathbf{0}$ *such that every* $\mathbf{0} \leq \mathbf{b} \leq \mathbf{a}$ *is in* **LFS***.*

Proof. By a result of Haught [9], if $\mathbf{a}$ is a Δ^0_2 1-generic degree, every $\mathbf{b} > \mathbf{0}$ below $\mathbf{a}$ is of 1-generic degree. Then the result follows immediately from Theorem 12. ◀

Another interesting corollary is that every non-computable c.e. set bounds a non-computable low for speed set. Likewise almost every set, in the measure-theoretic sense, *bounds* a non-computable low for speed set.

▶ **Corollary 14.** *Every non-zero c.e. degree bounds a member* $\mathbf{LFS}^*$*, every* 2*-random degree bounds a member of* $\mathbf{LFS}^*$*.*

Proof. This is simply because every non-zero c.e. degree and every 2-random degree bounds a 1-generic degree [13, 11]. ◀

We now show that **LFS** avoids a cone, namely all degrees above $\mathbf{0}'$.

▶ **Theorem 15.** *If* $\mathbf{a} \geq \mathbf{0}'$*, then* $\mathbf{a} \notin \mathbf{LFS}$*.*

Proof. The proof of this theorem relies on the proof of a classical computational complexity theorem, namely Blum's speed-up theorem [6] (see also [12, Theorem 32.2]), which asserts that for every sufficiently fast growing computable function f, there exists a computable set R which admits no fastest algorithm in that for every i such that $\Phi_i = R$, there is a j such that $\Phi_j = R$ and $f(time(\Phi_j, x)) \leq time(\Phi_i, x)$ for almost every x.

Blum's theorem is proven as follows. We build R by diagonalization against all Φ_i, where for all x in order we try to find an 'active' $i \leq |x|$ such that $\Phi_i(x)$ converges in less than $f^{|x|-i}(|x|)$ steps, and if such an i is found, we set $R(x) = 1 - \Phi_i(x)$ for the smallest such i, and declare i inactive from that point on (since we have already ensured $\Phi_i \neq R$, we no longer need to deal with Φ_i). If no such i is found, set $R(x) = 0$. By construction, R is computable, and any Φ_i computing it must satisfy $time(\Phi_i, x) \geq f^{|x|-i}(|x|)$ for almost all $|x| \geq i$ (otherwise, Φ_i would be diagonalized at some point, because for any x such that $time(\Phi_i, x) < f^{|x|-i}(|x|)$, the only way Φ_i can escape diagonalization is when some other Φ_j with $j < i$ is diagonalized in priority, but this situation can happen at most i times). Now, suppose Φ_e is a functional that computes R. We need to show that there is another functional which computes R much faster than Φ_e. Fix an integer k and assume we are given as 'advice' the finite list σ_k of indices $i < k$ such that Φ_i eventually gets diagonalized against (and therefore i becomes inactive) in the construction of R. Now, we can compute R via the following procedure. In a first phase, simply follow the construction of R as described above, until we reach a point where all $i \in \sigma_k$ have become inactive. At this point, we know (only because we know σ_k!) that none of the $\{\Phi_i \mid i < k\}$ are relevant for the construction of R on future x. Thus, we enter a second phase where to compute each $R(x)$, we only need to simulate, for $k \leq j \leq |x|$, $\Phi_j(x)$ during $f^{|x|-j}(|x|)$ steps of computation. By dovetailing, this can be done in $poly(|x| \cdot f^{|x|-k}(|x|))$ (the polynomial being independent of k) which, if f is fast growing enough and k large enough compared to e, is $< f\big(f^{|x|-e}(|x|)\big)$, which in turn is $< f(time(\Phi_e, x))$ for almost all x (note that such a k can be computed uniformly given e).

Now, suppose we are given $A \geq_T \emptyset'$. Let $f(n) = 2^n$ and R the corresponding set in Blum's theorem. Note that in the above, determining whether a given i will eventually become inactive can be done uniformly relative to $\emptyset'$, and thus the list σ_k can be computed uniformly in k relative to $\emptyset'$. Thus, using A as oracle, we can consider the procedure Ψ^A which for each pair (Φ_e, p_e) of a functional and a polynomial sequentially, finds the k and σ_k above, checks whether $e \in \sigma_k$, in which case there is nothing to do as $\Phi_e \neq R$ by definition, and if not, use the above 2-phase method to compute R, until a large x is found such that $f(time(\Psi^A, x)) < time(\Phi_e, x)$, which for x sufficiently large guarantees $p_e(time(\Psi^A, x)) < time(\Phi_e, x)$, and we can then move on to the next index $e + 1$. ◀

▶ **Theorem 16.** *There are* $\mathbf{a}, \mathbf{b} \in \mathbf{LFS}$ *such that* $\mathbf{a} \vee \mathbf{b} \notin \mathbf{LFS}$

Proof. Let G_0 be 2-generic, i.e., 1-generic relative to $\emptyset'$. Consider $G_1 = G_0 \Delta \emptyset'$ where Δ is the symmetric difference. It is easy to check that G_1 is also 2-generic. Thus S_{G_0} and S_{G_1} (defined as in the proof of Theorem 12) are both low for speed (Theorem 12) but $S_{G_0} \oplus S_{G_1} \geq_T G_0 \oplus G_1 \geq_T G_0 \Delta G_1 = \emptyset'$, so by the previous theorem, $deg(S_{G_0} \oplus S_{G_1}) \notin \mathbf{LFS}$. ◀

Using a diagonalization technique like in Blum's theorem (though with the same time bound for all functionals), we can prove that LFS has measure 0. In fact, we get a more precise statement in terms of algorithmic randomness.

▶ **Theorem 17.** *No Schnorr random sequence A is low for speed.*

Proof. The extra ingredient we need on top of Blum's theorem is to make the set R sparse: it will contains at most one string of each length. For each n, we compute $\Phi_i(x)$ during $f(|x|) = 2^{|x|}$ of computation for all x of length n and all active $i \leq n$ (here we don't need to have different time bounds for different functionals; all we need is for f to be computable and sufficiently fast-growing). If for any of these strings we see that $\Phi_i(x)$ converges, we take the smallest such i and then the smallest x for which we see convergence, and set $R(x) = 1 - \Phi_i(x)$, as well as $R(y) = 0$ for all $y \neq x$ of length n, and then declare i inactive. This way we do ensure sparseness of R, and like in the previous proof, that for all i, if Φ_i computes R, $time(\Phi_i, x) > 2^{|x|}$ for almost all x. On the other hand, consider the following procedure Ψ. Given oracle Z and input x, $\Psi^Z(x)$ first splits Z (viewed as a binary sequence) as $Z = \zeta_1\zeta_2\ldots$ with $|\zeta_i| = i$ and $\Psi^Z(x)$ returns 0 if $x = \zeta_{|x|}$ (thus the resulting computation is polynomial in $|x|$), and $\Psi^Z(x) = R(x)$ otherwise, using a fixed procedure to compute R. So there is a polynomial p such that for any Z, $time(\Psi^Z, x) \leq p(|x|)$ for infinitely many x's. Furthermore, we can only have $\Psi^Z(x) \neq R(x)$ if $x = \zeta_{|x|}$ and $\zeta_{|x|}$ happens to be the only string of its length in R. This has probability at most $2^{-|x|}$ ('at most' because R can also have no string of length $|x|$ at all) if Z is chosen at random. This means that, by setting $C_n = \{Z \mid (\exists x)\, |x| = n \wedge \Psi^Z(x) \neq R(x)\}$, we have $\lambda(C_n) \leq 2^{-n}$.

The C_n's are uniformly computable clopen subsets of $\{0,1\}^\omega$ because Ψ is a truth-table functional. Thus, a Schnorr random A can only belong to finitely many C_n's (see for example [5, Lemma 1.5.9]), meaning that $\Psi^A(x) = R(x)$ for almost all x. Thus there is a finite variation $\hat{\Psi}$ of Ψ such that $\hat{\Psi}^A = R$, and $\hat{\Psi}^A(x)$ is computed in polynomial time for infinitely many x while $time(\Phi_i, x) > 2^{|x|}$ for any Φ_i computing R and almost all x. This shows that A is not low for speed. ◀

Theorem 17 shows that LFS is a nullset, but it leaves open the possibility that almost all X are *Turing-equivalent* to a low for speed set. This would be similar to the category situation where – under the reasonable assumption $\mathsf{P} \neq \mathsf{NP}$ – the set LFS is meager (as proven by Bayer and Slaman) but the set of A's whose *degree* is in **LFS** is co-meager (Theorem 12). It turns out that **LFS** behaves quite differently in the measure setting:

▶ **Theorem 18.** *The set* $\{A \in \{0,1\}^\omega \mid deg(A) \in \mathbf{LFS}\}$ *has measure* 0.

At this point, we know that the set of X's which compute a member of LFS^* is very large: it has measure 1 and is co-meager, it contains every c.e. set, etc. We might even start thinking that *every* non-computable X computes a member of LFS^*. This is not the case however, as shown by the following theorem (which contrasts Corollary 13).

▶ **Theorem 19.** *There is a degree* $\mathbf{a} > \mathbf{0}$ *such that no* $\mathbf{0} < \mathbf{b} \leq \mathbf{a}$ *is in* **LFS**. *Indeed,* $\mathbf{a}$ *can be chosen to be a minimal Turing degree.*

The classical construction of a minimal degree is done by forcing over total computable function trees (here we follow the terminology of [16]). A function tree is a partial function $T : \{0,1\}^* \to \{0,1\}^*$ such that if either $T(\sigma 0)$ or $T(\sigma 1)$ is defined, then all of $T(\sigma)$, $T(\sigma 0)$ and $T(\sigma 1)$ are, and $T(\sigma 0)$ and $T(\sigma 1)$ are strict extensions of $T(\sigma)$ such that $T(\sigma 0) \perp T(\sigma 1)$ (we say that $T(\sigma 0)$ and $T(\sigma 1)$ split $T(\sigma)$). We say that σ is a node of T if $\sigma \in rng(T)$. A tree S is a sub-f-tree of T, which we denote by $S \preccurlyeq T$ when every node of S is a node of T. An infinite binary sequence Z is a path on an f-tree T if infinitely many prefixes of Z are nodes of T. The set of paths of T is denoted by $[T]$. An f-tree naturally extends to a functional from $\{0,1\}^\omega$ to $\{0,1\}^* \cup \{0,1\}^\omega$ by setting $T(X) = \bigcup_{\sigma \preccurlyeq X} T(\sigma)$ When an f-tree T is total, this extension is an homeomorphism from $\{0,1\}^\omega$ to $\{0,1\}^\omega$, and if T is furthermore computable, its inverse T^{-1} is also computable. Given a functional Φ_e, we say

that T is e-consistent if for any two nodes σ and τ on T and any n, if $\Phi_e^\sigma(n)$ and $\Phi_e^\tau(n)$ are both defined, then they are equal; in this case, for any path X of T, Φ_e^X is either partial or computable. We say that T is e-splitting if T is total and for any σ, $\Phi_e^{T(\sigma 0)}$ and $\Phi_e^{T(\sigma 1)}$ are incomparable; in this case, the restriction of Φ_e to $[T]$ is total, one-to-one, and its inverse is computable.

The key lemma in the construction of a minimal degree states that for any total computable f-tree T and every e, there is a total computable sub-f-tree S of T which is either e-consistent or e-splitting. Then an $X \in \{0,1\}^\omega$ of minimal degree is obtained by taking a sufficiently generic filter G over the set of total computable f-trees ordered by $\preccurlyeq$, and take the intersection of their sets of paths (the non-computability of X can be further ensured by remarking that for any σ, the set of computable f-trees whose nodes are all incomparable with σ is a dense set for the order $\preccurlyeq$, thus one can choose X to avoid any fixed subset of $\{0,1\}^\omega$, such as its computable elements).

We are going to prove Theorem 19 by showing that taking a sufficiently generic filter for the order $\preccurlyeq$ ensures lowness for speed as well. For this, we will make use of the following lemma.

▶ **Lemma 20.** *Let T be a total computable f-tree. There exists a total computable f-subtree $S \preccurlyeq T$ none of whose paths is low for speed.*

Proof. The functional $T^{-1} : [T] \to \{0,1\}^\omega$ is total on its domain, which is a Π_1^0 class, and thus is a tt-reduction by effective compactness. Let f be a computable time bound for the running time of T^{-1}, that is, for any $Y \in [T]$, whether $x \in T^{-1}(Y)$ can be decided in time $f(|x|)$ with access to oracle Y. Now, let L be a computable set that cannot be computed in time $2^{f(n+1)}$. Let S be the sub-f-tree of T defined by $S(\sigma) = T(\sigma \oplus L)$ (where $\sigma \oplus L = \sigma(0)L(0)\dots\sigma(k-1)L(k-1)$ when $k = |\sigma|$). The paths of S are exactly the sets of the form $T(X \oplus L)$ for some X. Each of them computes L in time $f(n+1)$ by definition of f, which is exponentially faster than any procedure computing L without oracle by our assumption on L. ◀

Proof of Theorem 19. Let T be a total computable f-tree and Φ_e a functional. As we explained earlier, the usual construction of a minimal degree shows that there is $S \preccurlyeq T$ which is either e-consistent or e-splitting. In the case S is e-consistent, we are satisfied (this guarantees Φ_e^A to be either partial or computable). If it is e-splitting, we further refine S as follows. Since S is e-splitting, we consider the total computable f-tree S' corresponding to the image of S by Φ_e: $S'(\sigma) = \Phi_e^{S(\sigma)}$ (this is indeed an f-tree precisely because S is e-splitting). By the previous lemma, there is a total computable $S'' \preccurlyeq S'$ none of whose paths is low for speed. Now the pullback $T' = \Phi_e^{-1}(S'')$ is a total computable f-subtree of T, which forces Φ_e^A to not be low for speed.

Thus we can force for all e that Φ_e^A is partial or not low for speed, and force A to be of minimal degree and be non-computable as usual. ◀

Our last theorem shows that the analogue of the low basis theorem (which asserts that every non-empty Π_1^0 class contains a member of low Turing degree) for lowness for speed fails.

▶ **Theorem 21.** *If $\mathbf{a}$ is a PA-degree, $\mathbf{a} \notin \mathbf{LFS}$. Thus there is a non-empty Π_1^0 class such that no member has a low for speed degree.*

Note that this is in fact a stronger statement than Theorem 15 since $\mathbf{0'}$ is a PA-degree.

References

1. Eric Allender, Harry Buhrman, and Michal Koucký. What can be efficiently reduced to the Kolmogorov-random strings? *Annals of Pure and Applied Logic*, 138:2–19, 2006.
2. Eric Allender, Luke Friedman, and William I. Gasarch. Limits on the computational power of random strings. *Information and Computation*, 222:80–92, 2013.
3. Theodore Baker, John Gill, and Robert Solovay. Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question. *SIAM Journal on Computing*, 4(4):431–442, 1975.
4. Robertson Bayer. *Lowness For Computational Speed*. PhD thesis, University of California Berkeley, 2012.
5. Laurent Bienvenu. *Game-theoretic characterizations of randomness: unpredictability and stochasticity*. PhD thesis, Université de Provence, 2008. https://tel.archives-ouvertes.fr/tel-00332425v2.
6. Manuel Blum. On effective procedures for speeding up algorithms. *Journal of the ACM*, 18(290-305), 1971.
7. Mingzhong Cai, Rodney Downey, Rachel Epstein, Steffen Lempp, and Joseph Miller. Random strings and tt-degrees of Turing complete c.e. sets. *Logical Methods in Computer Science*, 10(3), 2014.
8. Rodney Downey and Denis Hirschfeldt. *Algorithmic randomness and complexity*. Theory and Applications of Computability. Springer, 2010.
9. Christine A. Haught. The degrees below a 1-generic degree $< 0'$. *Journal of Symbolic Logic*, 51, 1986.
10. Carl Jockusch. Degrees of generic sets. In Frank Drake and Stanley S. Wainer, editors, *Recursion theory: its generalizations and applications*, number 45 in London Mathematical Society Lecture Note Series, pages 110–139. Cambridge Unversity Press, 1980.
11. Steven M. Kautz. *Degrees of random sets*. PhD thesis, Cornell University, 1991.
12. Dexter Kozen. *Theory of Computation*. Springer, New York, 2006.
13. Stuart Kurtz. *Randomness and genericity in the degrees of unsolvability*. PhD dissertation, University of Illinois at Urbana, 1981.
14. André Nies. *Computability and randomness*. Oxford Logic Guides. Oxford University Press, 2009.
15. Gerald Sacks. On the degrees less than 0'. *Annals of Mathematics*, 77:211–231, 1963.
16. Robert Soare. *Turing Computability: Theory and Applications*. Theory and Applications of Computability. Springer, 2016.

Large Flocks of Small Birds: on the Minimal Size of Population Protocols

Michael Blondin
Technische Universität München, Munich, Germany
blondin@in.tum.de

Javier Esparza
Technische Universität München, Munich, Germany
esparza@in.tum.de

Stefan Jaax
Technische Universität München, Munich, Germany
jaax@in.tum.de

Abstract

Population protocols are a well established model of distributed computation by mobile finite-state agents with very limited storage. A classical result establishes that population protocols compute exactly predicates definable in Presburger arithmetic. We initiate the study of the minimal amount of memory required to compute a given predicate as a function of its size. We present results on the predicates $x \geq n$ for $n \in \mathbb{N}$, and more generally on the predicates corresponding to systems of linear inequalities. We show that they can be computed by protocols with $O(\log n)$ states (or, more generally, logarithmic in the coefficients of the predicate), and that, surprisingly, some families of predicates can be computed by protocols with $O(\log \log n)$ states. We give essentially matching lower bounds for the class of 1-aware protocols.

2012 ACM Subject Classification Theory of computation → Distributed computing models, Theory of computation → Complexity theory and logic, Theory of computation → Logic and verification

Keywords and phrases Population Protocols, Presburger Arithmetic

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.16

Related Version The full version of this paper can be found at `https://arxiv.org/abs/1801.00742`.

Funding M. Blondin was supported by the Fonds de recherche du Quebec – Nature et technologies (FRQNT).

1 Introduction

Population protocols [4] are a model of distributed computation by anonymous, identical, and mobile finite-state agents. Initially introduced to model networks of passively mobile sensors, they also capture the essence of distributed computation in trust propagation or chemical reactions, the latter under the name of chemical reaction networks (see e.g. [18]). Structurally, population protocols can also be seen as a special class of Petri nets or vector addition systems [11].

Since the agents executing a protocol are anonymous and identical, its global state – called a *configuration* – is completely determined by the number of agents at each local state. In each computation step, a pair of agents, chosen by an adversary subject to a fairness

condition stating that any repeatedly reachable configuration is eventually reached, interact and move to new states according to a joint transition function. In a closely related model, the adversary chooses the pair of agents uniformly at random.

A protocol computes a boolean value for a given initial configuration if in all fair executions all agents eventually agree to this value – so, intuitively, population protocols compute by reaching consensus. Given a set of initial configurations, the predicate computed by a protocol is the function that assigns to each configuration C the boolean value computed by the protocol starting from C.

Much research on population protocols has focused on their expressive power, i.e., the class of predicates computable by different classes of protocols (see e.g. [3, 6, 13, 16, 7]). In a famous result [6], Angluin et al. have shown that predicates computable by population protocols are exactly the predicates definable in Presburger arithmetic. There is also much work on complexity metrics for protocols. The main two metrics are the *runtime* of a protocol – defined for the model with a randomized adversary as the expected number of pairwise interactions until all agents have the correct output value – and its *state space size*, e.g. the number of states of each agent. In [5], Angluin et al. show that every Presburger predicate is computed with high probability by a population protocol with a leader – a distinguished auxiliary agent that assumes a specific state in the initial configuration irrespective of the input – in $O(n \log^4 n)$ interactions in expectation, where n is the number of agents of the initial configuration. Several recent papers study time-space trade-offs for specific tasks, like electing a leader [10], or for specific predicates, like majority [2, 1, 9].

In this paper we study the state space size of protocols as a function of the predicate they compute. In particular, we are interested in the minimal number of states needed to evaluate systems of linear constraints (a large subclass of the predicates computed by population protocols) as a function of the number of bits needed to describe the system. To the best of our knowledge, this question has not been considered so far. We study the question for protocols with and without leaders. Our results show that protocols with leaders can be exponentially more compact than leaderless protocols.

In order to introduce our results in the simplest possible setting, in the first part of the paper we focus on the family of predicates $\{x \geq n : n \in \mathbb{N}\}$. These predicates specify the well-known flock-of-birds problem [4], in which tiny sensors placed on birds have to reach consensus on whether the number of sick birds in a flock exceeds a given constant. The minimal number of states for computing $x \geq n$ formalizes a very natural question about emerging behavior: How many states must agents have in order to exhibit a "phase transition" when their number reaches n? The standard protocol for the predicate $x \geq n$ (see Example 1) has $n+1$ states. We are interested in protocols with at most $O(\log n)$ states, either leaderless or with at most $O(\log n)$ leaders. In the second part of the paper, we generalize our results to a much larger class of predicates, namely systems of linear inequalities $A\boldsymbol{x} \geq \boldsymbol{b}$. Since $x \geq n$ is a (very) special case, our lower bounds for flock-of-birds protocols apply, while the upper bounds require new (and involved) constructions.

Protocol size for the flock-of-birds problem. In a first warm-up phase we exhibit a family of leaderless protocols with only $O(\log n)$ states. More precisely, we prove:

(1) There exists a family $\{\mathcal{P}_n : n \in \mathbb{N}\}$ of leaderless population protocols such that $\mathcal{P}_n$ has $O(\log_2 n)$ states and computes the predicate $x \geq n$ for every $n \in \mathbb{N}$.

We also give a lower bound:

(2) For every family $\{\mathcal{P}_n : n \in \mathbb{N}\}$ of leaderless population protocols such that $\mathcal{P}_n$ computes $x \geq n$, there exist infinitely many n such that $\mathcal{P}_n$ has at least $(\log n)^{1/4}$ states.

However, this bound is only *existential* ("there exists infinitely many n" instead of "for all n"). Moreover, it follows from a counting argument that does not provide any information on the values of n realizing the bound. Is there a poly-logarithmic universal bound? We show that, surprisingly, the answer is negative:

(3) There exists a family $\{\mathcal{P}_n : n \in \mathbb{N}\}$ of population protocols with two leaders, and values $c_0 < c_1 < \ldots \in \mathbb{N}$, such that $\mathcal{P}_n$ has $O(\log \log c_n)$ states and computes the predicate $x \geq c_n$ for every $n \in \mathbb{N}$.

Observe that in these protocols the "phase transition" occurs at $x = c_n$, even though no agent has enough memory to index a particular bit of c_n.

Can one go even further, and design $O(\log \log \log c_n)$ protocols? We show that the answer is negative for *1-aware* protocols. Both the standard protocol for $x \geq n$ and the families of (1) and (3) have the following, natural property: If the number of agents is greater than or equal to n, then the agents not only reach consensus 1, they also eventually *know* that they will reach this consensus. We say that these protocols are 1-aware.

We obtain lower bounds for 1-aware protocols that essentially match the upper bounds of (1) and (3):

(4) Every leaderless, 1-aware population protocol computing $x \geq n$ has at least $\log_3 n$ states.

(5) Every 1-aware protocol (leaderless or not) computing $x \geq n$ has at least $(\log \log(n)/151)^{1/9}$ states.

Protocols for systems of linear inequalities. In the second part of the paper we show that our results can be extended to other predicates. First, instead of the simple predicate $x \geq n$, we study the general linear predicate $a_1x_1 + a_2x_2 + \cdots + a_kx_k \geq c$ for arbitrary integer coefficients $a_1, \ldots, a_k, c \in \mathbb{Z}$. By means of a delicate construction we give protocols whose number of states grows only logarithmically in the size of the coefficients:

(6) There is a protocol with at most $O(kn)$ states and $O(n)$ leaders that computes $a_1x_1 + \cdots + a_kx_k \geq c$, where n is the size of the binary encoding of $\max(|a_1|, |a_2|, \ldots, |a_k|, |c|)$.

Finally, in the most involved construction of the paper, we show that the same applies to arbitrary systems of linear inequalities. Note that the standard conjunction construction, which produces a protocol for $\varphi_1 \wedge \varphi_2$ from protocols computing predicates φ_1 and φ_2, cannot be applied because it would lead to exponentially large protocols.

(7) There is a protocol with at most $O((\log m + n)(m + k))$ states and $O(m(\log m + n))$ leaders that computes $A\boldsymbol{x} \geq \boldsymbol{c}$, where $A \in \mathbb{Z}^{m \times k}$ and n is the size of the largest entry in A and $\boldsymbol{c}$.

Structure of the paper. Section 2 introduces basic definitions, protocols with and without leaders, and a simple construction with an involved correctness proof showing how to simulate protocols with k-way interactions by standard protocols. Sections 3 to 5 present our bounds on the flock-of-birds predicates, and Section 6 the bounds on systems of linear inequalities. Due to space constraints, some proofs are deferred to the full version of this paper.

2 Preliminaries

Numbers. Let $n \in \mathbb{N}_{>0}$. The logarithm in base b of n is denoted by $\log_b n$. Whenever $b = 2$, we omit the subscript. We define $\text{bits}(n)$ as the set of indices of the bits occurring in the binary representation of n, e.g. $\text{bits}(13) = \{0, 2, 3\}$ since $13 = 1101_2$. The *size* of n, denoted $\text{size}(n)$, is the number of bits required to represent n in binary. Note that $|\text{bits}(n)| \leq \text{size}(n) = \lfloor \log n \rfloor + 1$.

Multisets. A *multiset* over a finite set E is a mapping $M\colon E \to \mathbb{N}$. The set of all multisets over E is denoted $\mathbb{N}^E$. For every $e \in E$, $M(e)$ denotes the number of occurrences of e in M, and for every $E' \subseteq E$ we define $M(E') \stackrel{\text{def}}{=} \sum_{e\in E'} M(e)$. The *support* and *size* of M are defined respectively as $[\![M]\!] \stackrel{\text{def}}{=} \{e \in E : M(e) > 0\}$ and $|M| \stackrel{\text{def}}{=} \sum_{e\in E} M(e)$. *Addition* and *comparison* are extended to multisets componentwise, i.e. $(M + M')(e) \stackrel{\text{def}}{=} M(e) + M'(e)$ for every $e \in E$, and $M \leq M' \stackrel{\text{def}}{\iff} M(e) \leq M(e)$ for every $e \in E$. We define *multiset difference* as $(M \ominus M')(e) \stackrel{\text{def}}{=} \max(M(e) - M'(e), 0)$ for every $e \in E$. The empty multiset is denoted $\mathbf{0}$. We sometimes denote multisets using a set-like notation, e.g. $\wr f, 2 \cdot g, h \wr$ is the multiset M such that $M(f) = 1$, $M(g) = 2$, $M(h) = 1$ and $M(e) = 0$ for every $e \in E \setminus \{f, g, h\}$.

Population protocols. We introduce a rather general model of population protocols, allowing for interactions between more than two agents and for leaders. A *k-way population protocol* is a tuple $\mathcal{P} = (Q, T, I, L, O)$ such that

- Q is a finite set of *states*,
- $T \subseteq \bigcup_{2\leq i\leq k} Q^i \times Q^i$ is a set of *transitions*,
- $I \subseteq Q$ is a set of *initial states*,
- $L \in \mathbb{N}^Q$ is a set of *leaders*, and
- $O\colon Q \to \{0, 1\}$ is the *output mapping*.

We assume throughout the paper that agents can always interact, i.e., that for every pair of states (p, q), there exists a pair of states (p', q') such that $((p, q), (p', q')) \in T$.

A *configuration* of $\mathcal{P}$ is a multiset $C \in \mathbb{N}^Q$ such that $|C| > 0$. Intuitively, C describes a non empty collection containing $C(q)$ agents in state q for every $q \in Q$. We denote the set of configurations over $E \subseteq Q$ by $\mathrm{Pop}(E)$. A configuration C is *initial* if $C = D + L$ for some $D \in \mathrm{Pop}(I)$. So, intuitively, leaders are distinguished agents that are present in every initial configuration. The *number of leaders* of $\mathcal{P}$ is $|L|$. We say that $\mathcal{P}$ is *leaderless* if it has no leader, i.e. if $L = \mathbf{0}$. We discuss protocols with and without leaders later in this section.

Let $t = ((p_1, p_2, \ldots, p_i), (q_1, q_2, \ldots, q_i))$ be a transition. To simplify the notation, we denote t as $p_1, p_2, \ldots, p_i \mapsto q_1, q_2, \ldots, q_i$. Intuitively, t describes that i agents at states $p_1, \ldots, p_i$ may interact and move to states $q_1, \ldots, q_i$. The *preset* and *postset* of t are respectively defined as ${}^\bullet t \stackrel{\text{def}}{=} \{p_1, p_2, \ldots, p_i\}$ and $t^\bullet \stackrel{\text{def}}{=} \{q_1, q_2, \ldots, q_i\}$. We extend presets and postsets to sets of transitions, e.g. ${}^\bullet T \stackrel{\text{def}}{=} \bigcup_{t\in T} {}^\bullet t$. The *pre-multiset* and *post-multiset* of t are respectively defined as $\mathrm{pre}(t) \stackrel{\text{def}}{=} \wr p_1, p_2, \ldots, p_i \wr$ and $\mathrm{post}(t) \stackrel{\text{def}}{=} \wr q_1, q_2, \ldots, q_i \wr$.

We say that t is *enabled* at $C \in \mathrm{Pop}(Q)$ if $C \geq \mathrm{pre}(t)$. If t is enabled at C, then it can *occur*, in which case it leads to the configuration $C' = (C \ominus \mathrm{pre}(t)) + \mathrm{post}(t))$. We denote this by $C \xrightarrow{t} C'$. We say that t is *silent* if $\mathrm{pre}(t) = \mathrm{post}(t)$. In particular, if t is silent and $C \xrightarrow{t} C'$, then $C = C'$. We write $C \to C'$ if $C \xrightarrow{t} C'$ for some $t \in T$. We write $C \xrightarrow{t_1 t_2 \cdots t_k} C'$ if there exist $C_0, C_1, \ldots, C_k \in \mathrm{Pop}(Q)$ and $t_1, t_2, \ldots, t_k \in T$ such that $C = C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} \cdots C_k = C'$. We write $C \xrightarrow{*} C'$ if $C \xrightarrow{\sigma} C'$ for some $\sigma \in T^*$. We say that C' is *reachable* from C if $C \xrightarrow{*} C'$. The *support* of a sequence $\sigma = t_1 t_2 \cdots t_n \in T^*$ is $[\![\sigma]\!] \stackrel{\text{def}}{=} \{t_i : 1 \leq i \leq n\}$.

▶ **Example 1.** The flock-of-birds protocol mentioned in the introduction is formally defined as $\mathcal{P}_n = (Q, T, I, L, O)$ where $Q = \{0, 1, \ldots, n\}$, $I = \{1\}$, $L = \mathbf{0}$, $O(a) = 1 \iff a = n$, and where T consists of the following transitions:

$$s_{a,b} : a, b \mapsto 0, \min(a + b, n) \qquad \text{for every } 0 \leq a, b < n,$$
$$t_a : a, n \mapsto n, n \qquad \text{for every } 0 \leq a \leq n.$$

$\mathcal{P}_n$ is 2-way and leaderless. Intuitively, it works as follows. Each agent stores a number. When two agents meet, one agent stores the sum of their values and the other one stores

0. Sums cap at n. Once an agent reaches n, all agents eventually get converted to n. To illustrate the above definitions, observe that: ${}^\bullet s_{2,3} = \{2,3\}$, $t_2^\bullet = \{n\}$, $\mathrm{pre}(s_{2,3}) = \multiset{2,3}$ and $\mathrm{post}(t_2) = \multiset{n,n}$. Configuration $\multiset{1,1,1}$ is initial, but $\multiset{1,0,2}$ is not. We have $\multiset{1,1,1} \xrightarrow{s_{1,1}} \multiset{1,0,2} \xrightarrow{t_0} \multiset{1,2,2} \xrightarrow{t_1} \multiset{2,2,2}$, or more concisely $\multiset{1,1,1} \xrightarrow{\sigma} \multiset{2,2,2}$ where $\sigma = s_{1,1}t_0t_1$.

Computing with population protocols. An *execution* π is an infinite sequence of configurations $C_0C_1\cdots$ such that $C_0 \to C_1 \to \cdots$. We say that π is *fair* if for every configuration D the following holds[1]:

if $\{i \in \mathbb{N} : C_i \xrightarrow{*} D\}$ is infinite, then $\{i \in \mathbb{N} : C_i = D\}$ is infinite.

In other words, fairness ensures that a configuration cannot be avoided forever if it can be reached infinitely often along π. We say that a configuration C is a *consensus configuration* if $O(p) = O(q)$ for every $p, q \in [\![C]\!]$. If a configuration C is a consensus configuration, then its *output* $O(C)$ is the unique output of its states, otherwise it is $\bot$. An execution $\pi = C_0C_1\cdots$ *stabilizes* to $b \in \{0,1\}$ if $O(C_i) = O(C_{i+1}) = \cdots = b$ for some $i \in \mathbb{N}$. The *output* of π is $O(\pi) \stackrel{\text{def}}{=} b$ if it stabilizes to b, and $O(\pi) \stackrel{\text{def}}{=} \bot$ otherwise. A consensus configuration C is *stable* if every configuration C' reachable from C is a consensus configuration such that $O(C') = O(C)$. It can easily be shown that a fair execution stabilizes to $b \in \{0,1\}$ if and only if it contains a stable configuration whose output is b.

A population protocol $\mathcal{P} = (Q, T, I, L, O)$ is *well-specified* if for every initial configuration C_0, there exists $b \in \{0,1\}$ such that every fair execution π starting at C_0 has output b. If $\mathcal{P}$ is well-specified, then we say that it *computes* the predicate $\varphi\colon \mathrm{Pop}(I) \to \{0,1\}$ if for every $D \in \mathrm{Pop}(I)$, every fair execution starting at $D + L$ has output $\varphi(D)$.

▶ **Example 2.** Consider the protocol $\mathcal{P}_2$ defined in Example 1 (i.e, $n = 2$). We have $O(\multiset{1,1,1}) = 0$, $O(\multiset{2,2,2}) = 1$ and $O(\multiset{1,0,2}) = \bot$. The execution $\multiset{1,1,1} \to \multiset{1,0,2} \to \multiset{1,2,2} \to \multiset{2,2,2} \to \multiset{2,2,2} \to \cdots$ is fair and its output is 1. However, the execution $\multiset{1,1,1} \to \multiset{1,0,2} \to \multiset{1,0,2} \to \cdots$ is not fair since $\multiset{1,0,2}$ occurs infinitely often and can lead to $\multiset{2,2,2}$ which does not occur.

Leaders. Intuitively, leaders are extra agents present in every initial configuration. Allowing a large number of leaders may help to compute predicates with fewer states. To illustrate this, consider the leaderless protocol of Example 1. It computes $x \geq n$ with $n+1$ states. We describe a 2-way protocol with only 4 states, but n leaders. It is an adaptation of the well-known basic majority protocol (see, e.g., [8]). Let $\mathcal{P}'_n = (Q, T, I, L_n, O)$ be the protocol where $Q \stackrel{\text{def}}{=} \{x, y, \overline{x}, \overline{y}\}$, $I \stackrel{\text{def}}{=} \{x\}$, $L_n \stackrel{\text{def}}{=} \multiset{n \cdot y}$, $O(x) = O(\overline{x}) \stackrel{\text{def}}{=} 1$, $O(y) = O(\overline{y}) \stackrel{\text{def}}{=} 0$, and where T consists of the following transitions:

$$x, y \mapsto \overline{x}, \overline{y}, \qquad x, \overline{y} \mapsto x, \overline{x}, \qquad y, \overline{x} \mapsto y, \overline{y}, \qquad \overline{x}, \overline{y} \mapsto \overline{x}, \overline{x}.$$

Informally, "active" agents in states x and y collide and become "passive" agents in states $\overline{x}$ and $\overline{y}$. At some point, some active agents "win" and convert all passive agents to their output. It is known that this protocol is well-specified and computes the predicate $x \geq y$ when there are no leaders (i.e., if we set $L_n = \mathbf{0}$). So, by initially fixing n leaders in state y, $\mathcal{P}'_n$ computes $x \geq n$.

[1] This definition of fairness differs from the original definition of Angluin et al. [4], but is equivalent.

Thus, the predicate $x \geq n$ can be computed either with $O(n)$ states and no leaders, or with 4 states and $O(n)$ leaders. This indicates a trade-off between states and leaders, and one should avoid hiding all of the complexity in one of them. For this reason, we make these two quantities explicit in all of our results.

The reason for considering protocols with leaders is that, as we shall see, even a constant number of leaders demonstrably leads to exponentially more compact protocols for some predicates. Other papers have made similar observations with respect to other resource measures (see e.g. [5, 14]).

From k-way to 2-way protocols. In our constructions it is very convenient to use k-way transitions for $k > 2$. The following lemma shows that k-way protocols can be transformed into 2-way protocols by introducing a few extra states. Intuitively, a k-way transition is simulated by a chain of 2-way transitions. The first part of the chain "collects" k participants one by one. First, two agents agree to participate, and one of them becomes "passive", while the second "searches" for a third participant. This is iterated until k participants are collected. In the second part, the last collected agent "informs" all passive agents, one by one, that k agents have been collected; upon hearing this, the passive agents move to their destination states and become active again. To prevent faulty behavior when there are not enough agents, all transitions of the first part can be "reversed", that is, the agent that is currently searching and the last collected agent can "repent" and "undo" the transition. While the construction is simple and intuitive, its correctness proof is very involved, because agents that reach their destination can engage in other interactions while other participants are still passive. The construction is presented in the full version of this paper.

▶ **Lemma 3.** *Let $\mathcal{P} = (Q, T, I, L, O)$ be a well-specified k-way population protocol. For every $3 \leq i \leq k$, let n_i be the number of i-way transitions of $\mathcal{P}$. There exists a 2-way population protocol $\mathcal{P}'$, with at most $|Q| + \sum_{3 \leq i \leq k} 3i \cdot n_i$ states, which is well-specified and computes the same predicate as $\mathcal{P}$.*

3 Leaderless protocols for $x \geq n$

In this section, we consider *leaderless* protocols for the predicate $x \geq n$. We first show that the number of states required to compute this predicate can be reduced from the known $O(n)$ bound to $O(\log n)$, using a similar binary encoding as in [1]. Then we show an existential lower bound of $O((\log n)^{1/4})$.

A protocol with $O(\log n)$ states. We describe a leaderless size(n)-way protocol $\mathcal{P}_n = (Q_n, T_n, I_n, \mathbf{0}, O_n)$ with size$(n) + 3$ states that computes $x \geq n$. The states are $Q_n \stackrel{\text{def}}{=} \{\mathbf{0}, \mathbf{2^0}, \ldots, \mathbf{2^{size(n)}}, \mathbf{n}\}$ and the sole initial state is $I_n \stackrel{\text{def}}{=} \{\mathbf{2^0}\}$. The output mapping is defined as $O_n(\mathbf{n}) \stackrel{\text{def}}{=} 1$ and $O_n(q) \stackrel{\text{def}}{=} 0$ for every state $q \neq \mathbf{n}$.

Before defining the set T_n of transitions, we need some preliminaries. For every state $q \in Q_n$, let val(q) denote the number q stands for, i.e. val$(\mathbf{0}) = 0$, val$(\mathbf{n}) = n$ and val$(\mathbf{2^i}) = 2^i$ for every $0 \leq i \leq$ size(n). Moreover, for every configuration C, let val$(C) \stackrel{\text{def}}{=} \sum_{q \in Q_n} \text{val}(q) \cdot C(q)$. A configuration C is a *representation of* m if val$(C) = m$. For example, the configuration $\{\!|\mathbf{0}, \mathbf{2^1}, 5 \cdot \mathbf{2^3}|\!\}$ is a representation of $0 + 2^1 + 5 \cdot 2^3 = 42$. Observe that every initial configuration C_0 is a representation of $|C_0|$.

T_n is the union of two sets T_n^1 and T_n^2. Intuitively, T_n^1 allows the protocol to reach from a representation of a number, say m, other representations of m. Formally, the transitions of

T_n^1 are:

$$\begin{array}{rcll} \mathbf{2^i}, \mathbf{2^i} & \mapsto & \mathbf{2^{i+1}}, \mathbf{0} & \text{for every } 0 \leq i < \text{size}(n) \\ \mathbf{2^{i+1}}, \mathbf{0} & \mapsto & \mathbf{2^i}, \mathbf{2^i} & \text{for every } 0 \leq i < \text{size}(n) \\ \wr \mathbf{2^i} : i \in \text{bits}(n) \wr & \mapsto & \mathbf{n}, \underbrace{\mathbf{0}, \cdots, \mathbf{0}}_{|\text{bits}(n)|-1 \text{ copies}} & \end{array}$$

The transitions of T_n^2 allow agents in state $\mathbf{n}$ to "attract" all other agents to $\mathbf{n}$. Formally, they are:

$$\mathbf{n}, q \quad \mapsto \quad \mathbf{n}, \mathbf{n} \quad \text{for every } q \in Q_n.$$

Let us show that $\mathcal{P}_n$ computes $x \geq n$. Let $C_0 = \wr m \cdot \mathbf{2^0} \wr$. If $m < n$, then $C(\mathbf{n}) = 0$ holds for every representation C of m. Therefore, every configuration C reachable from C_0 satisfies $C(\mathbf{n}) = 0$ and, since $\mathbf{n}$ is the only state with output 1, the protocol stabilizes to 0. If $m \geq n$, then it is possible to reach a representation C of m satisfying $C(\mathbf{n}) > 0$, for example $C = \wr \mathbf{n}, (m-n) \cdot \mathbf{2^0} \wr$. Since for every transition $\mathbf{2^i}, \mathbf{2^i} \mapsto \mathbf{2^{i+1}}, \mathbf{0}$ the set T_n also contains the reverse transition $\mathbf{2^{i+1}}, \mathbf{0} \mapsto \mathbf{2^i}, \mathbf{2^i}$, every representation C of m satisfying $C(\mathbf{n}) = 0$ can reach a representation C' of m satisfying $C'(\mathbf{n}) > 0$. Let $\pi = C_0 C_1 C_2 \cdots$ be a fair execution. By fairness, there is some $i \in \mathbb{N}$ such that $C_i(\mathbf{n}) > 0$. Again by fairness, and because of T_n^2, there is also an index j such that $C_k = \wr m \cdot \mathbf{n} \wr$ for every $k \geq j$, and so π stabilizes to 1.

Note that $|Q_n| = \text{size}(n) + 3$. Moreover, $\mathcal{P}_n$ has one $|\text{bits}(n)|$-way transition. Thus, by Lemma 3, we obtain the following theorem:

▶ **Theorem 4.** *There exists a family $\{\mathcal{P}_0, \mathcal{P}_1, \ldots\}$ of leaderless and 2-way population protocols such that $\mathcal{P}_n$ has at most $4\lfloor \log n \rfloor + 7$ states and computes the predicate $x \geq n$.*

An existential $(\log n)^{1/4}$ lower bound. We show that every family $\{\mathcal{P}_n\}_{n \in \mathbb{N}}$ of leaderless and 2-way protocols computing the family of predicates $\{x \geq n\}_{n \in \mathbb{N}}$ must contain infinitely many members of size $\Omega((\log n)^{1/4})$. We call this an existential lower bound, contrary to a universal lower bound, which would state that $\mathcal{P}_n$ has size $\Omega((\log n)^{1/4})$ for every $n \geq 1$.

▶ **Theorem 5.** *Let $\{\mathcal{P}_0, \mathcal{P}_1, \ldots\}$ be an infinite family of leaderless and 2-way population protocols such that $\mathcal{P}_n$ computes the predicate $x \geq n$ for every $n \in \mathbb{N}$. There exist infinitely many indices n such that $\mathcal{P}_n$ has at least $(\log n)^{1/4}$ states.*

Proof sketch. The proof boils down to bounding the number $d(m)$ of unary predicates computed by protocols with m states. The number of distinct sets of transitions, excluding silent ones, is bounded by $2^{m^4 - m^2}$. The number of possible initial states and output mappings are respectively m and 2^m. Altogether, we obtain:

$$d(m) \leq 2^{m^4-m^2} \cdot m \cdot 2^m = 2^{m^4} \cdot \frac{2^m \cdot m}{2^{m^2}} \leq 2^{m^4}.$$ ◀

4 A $O(\log \log n)$ protocol with leaders for some $x \geq n$

The lower bound of Section 3 is not valid for every n, it only ensures that, for some values of n, protocols computing $x \geq n$ must have a logarithmic number of states. We prove that, surprisingly, there is an infinite sequence $n_1 < n_2 < \cdots$ of values that break through the logarithmic barrier: The predicates $x \geq n_i$ can be computed by protocols with only $O(\log \log n_i)$ states and two leaders. So, loosely speaking, a flock of birds can decide if it contains at least n_i birds, even though no bird has enough memory to index a bit of n_i.

The result is based on a construction of [15]. In this paper, Mayr and Meyer study the word problem for commutative semigroup presentations. Given a finite set $\mathcal{A}$ of generators, a presentation of a commutative semigroup generated by $\mathcal{A}$ is a finite set of productions $\mathcal{S} = \{l_1 \to r_1, \ldots, l_m \to r_m\}$, where $l_i, r_i \in \mathcal{A}^*$ for every $1 \leq i \leq m$, satisfying:

- Commutativity: $ab \to ba \in \mathcal{S}$ for every $a, b \in \mathcal{A}$;[2] and
- Reversibility: if $l \to r \in \mathcal{S}$, then $r \to l \in \mathcal{S}$.

Given $\alpha, \beta \in \mathcal{A}^*$, we say that β is *derived* from α *in one step*, denoted by $\alpha \to \beta$, if $\alpha = \gamma\, l\, \delta$ and $\beta = \gamma\, r\, \delta$ for some $\gamma, \delta \in \mathcal{A}^*$ and some $r \to l \in \mathcal{S}$. We say that β is *derived* from α if $\alpha \xrightarrow{*} \beta$, where $\xrightarrow{*}$ is the reflexive transitive closure of the relation induced by $\to$. Observe that, by reversibility, we have $\alpha \xrightarrow{*} \beta$ iff $\beta \xrightarrow{*} \alpha$. Further, by commutativity we have $\alpha \xrightarrow{*} \beta$ iff $\pi(\alpha) \xrightarrow{*} \pi'(\beta)$ for every permutation π of $\mathcal{A}$.

Mayr and Meyer study the following question: given a commutative semigroup presentation $\mathcal{S}$ over $\mathcal{A}$, and initial and final letters $s, f \in \mathcal{A}$, what is the length of the shortest word α such that $s \xrightarrow{*} f\alpha$? They exhibit a family of presentations of size $O(n)$ for which the shortest α has double exponential length 2^{2^n}. More precisely, in [15, Sect. 6], they construct a family $\{\mathcal{S}_n\}_{n\geq 1}$ of presentations over alphabets $\{\mathcal{A}_n\}_{n\geq 1}$ satisfying the following properties:

(1) $|\mathcal{A}_n| = 14n + 10$, $|\mathcal{S}_n| = 20n + 8$, and $\max\{|l|, |r| : l \to r \in \mathcal{S}_n\} = 5$.
(2) $\{s_n, f_n, b_n, c_n\} \subseteq \mathcal{A}_n$ for every $n \geq 1$.
(3) $s_n c_n \xrightarrow{*} f_n \alpha$ iff $\alpha = c_n b_n^{2^{2^n}}$ [15, Lemma 6 and 8].

To apply this result, for each $n \geq 1$ we construct a 5-way population protocol $\mathcal{P}_n = (Q_n, T_n, I_n, L_n, O_n)$ with two leaders as follows:

- $Q_n \stackrel{\text{def}}{=} \mathcal{A}_n \cup \{x\}$ for some $x \notin \mathcal{A}_n$.
- $T_n \stackrel{\text{def}}{=} T_n^1 \cup T_n^2$, where:
  - T_n^1 contains a transition $\text{pad}(p)$ for every production $p = l \to r$ of $\mathcal{S}_n$, obtained by "padding" p with x so that its left and right sides have the same length. For example, $\text{pad}(aab \to cd) = a, a, b \mapsto c, d, x$, and $\text{pad}(a \to bc) = a, x \mapsto b, c$,
  - $T_n^2 \stackrel{\text{def}}{=} \{f_n, q \mapsto f_n, f_n \mid q \in Q_n\}$,
- $I_n \stackrel{\text{def}}{=} \{x\}$,
- $L_n \stackrel{\text{def}}{=} \{\!\{c_n, s_n\}\!\}$, and
- $O_n(f_n) \stackrel{\text{def}}{=} 1$ and $O_n(q) \stackrel{\text{def}}{=} 0$ for every $q \neq f_n$.

Intuitively, T_n^1 allows $\mathcal{P}_n$ to simulate derivations of $\mathcal{S}_n$: a step $C \xrightarrow{\text{pad}(p)} C'$ of $\mathcal{P}_n$ simulates a one-step derivation of $\mathcal{S}_n$. We make this more precise. Given $\alpha \in \mathcal{A}_n^*$ and $m \geq |\alpha|$, let $C_{\alpha,m}$ be the configuration of $\mathcal{P}_n$ defined as follows: $C_{\alpha,m}(x) = m$, and $C_{\alpha,m}(a) = |\alpha|_a$ for every $a \in \mathcal{A}_n$, where $|\alpha|_a$ is the number of occurrences of a in α. Further, given a configuration C of $\mathcal{P}_n$, let α_C be the element of $\mathcal{S}_n$ given by $\alpha_C = a_1^{C(a_1)} \cdots a_m^{C(a_m)}$, where $a_1, \ldots, a_m$ is a fixed enumeration of $\mathcal{A}_n$. We have:

▶ **Lemma 6.** *Let $\alpha, \beta \in \mathcal{A}_n^*$ and let C, C' be configurations of $\mathcal{P}_n$.*

(a) *If $\alpha \xrightarrow{p_1 \cdots p_k} \beta$ in $\mathcal{S}_n$, then for every $m \geq 4k$, $C_{\alpha,m} \xrightarrow{\text{pad}(p_1)\cdots\text{pad}(p_k)} C_{\beta,m'}$ in $\mathcal{P}_n$ for some $m' \geq 0$.*
(b) *If $C \xrightarrow{\text{pad}(p_1)\cdots\text{pad}(p_k)} C'$ in $\mathcal{P}_n$, then $\alpha_C \xrightarrow{p_1 \cdots p_k} \alpha_{C'}$ in $\mathcal{S}_n$.*

From Lemma 6, (1) and (3), the following can be shown:

▶ **Theorem 7.** *For every $n \in \mathbb{N}$, there is a 5-way protocol $\mathcal{P}_n$ with at most $14n+11$ states and at most $34n + 19$ transitions that computes the predicate $x \geq c_n$ for some number $c_n \geq 2^{2^n}$.*

[2] In [15], the elements of S are written using uppercase letters. We use lowercase for convenience.

Using Theorem 7 and Lemma 3, we obtain:

▶ **Corollary 8.** *There exists a family $\{\mathcal{P}_0, \mathcal{P}_1, \ldots\}$ of 2-way protocols with two leaders and a family $\{c_0, c_1, \ldots\}$ of natural numbers such that for every $n \in \mathbb{N}$ the following holds: $c_n \geq 2^{2^n}$ and protocol $\mathcal{P}_n$ has at most $314 \log\log c_n + 131$ states and computes the predicate $x \geq c_n$.*

5 Universal lower bounds for 1-aware protocols

To the best of our knowledge, all the protocols in the literature for predicates $x \geq n$, including those of Section 3 and Section 4, share a very natural property: if the number of agents is greater than or equal to n, then the agents not only eventually reach consensus 1, they also eventually *know* that they will reach this consensus. Let us formalize this idea:

▶ **Definition 9.** A well-specified population protocol $\mathcal{P} = (Q, T, I, L, O)$ is *1-aware* if there is a set $Q_1 \subseteq Q \setminus (I \cup [\![L]\!])$ of states such that for every initial configuration C_0 and every fair execution $\pi = C_0 C_1 \cdots$

(1) if π stabilizes to 0, then $C_i(Q_1) = 0$ for every $i \geq 0$, and

(2) if π stabilizes to 1, then there is some $i \geq 0$ such that $C_j(Q \setminus Q_1) = 0$ for every $j \geq i$.

If in the course of an execution π an agent reaches a state of Q_1, then π cannot stabilize to 0 by (1), and so, since $\mathcal{P}$ is well-specified, it stabilizes to 1; intuitively, at this moment the agent "knows" that the consensus will be 1. Further, if an execution stabilizes to 1, then all agents eventually reach and remain in Q_1 by (2), and so eventually all agents "know".[3] Albeit seemingly restrictive, 1-aware protocols compute a significant subclass of predicates: monotonic Presburger predicates (see the full version of the paper for more details).

We say that a state q is *coverable* from a configuration C if $C \xrightarrow{*} C'$ for some configuration C' such that $C'(q) > 0$. The fundamental property of 1-aware protocols is that, loosely speaking, consensus reduces to coverability:

▶ **Lemma 10.** *Let $\mathcal{P} = (Q, T, \{x\}, L, O)$ be a 1-aware protocol computing a unary predicate φ. We have $\varphi(n) = 1$ if and only if some state of Q_1 is coverable from $\wr n \cdot x \wr + L$.*

We show that for 1-aware protocols, the bounds of Sections 3 and 4 are essentially tight.

Leaderless protocols. We prove that a 1-aware, leaderless and 2-way protocol computing $x \geq n$ has at least $\log_3 n$ states. By Lemma 10, it suffices to show that some state of Q_1 is coverable from $\wr 3^k \cdot q \wr$, where q is the initial state. Proposition 11 below is the key to the proof. It states that for every finite execution $C_1 \xrightarrow{\pi} C_2$, there is $C_1' \xrightarrow{\pi'} C_2'$ such that C_1' has the same support as C_1 and is not too large, and C_2' contains a "record" of all states encountered during the execution of π (this is the set $[\![C_1]\!] \cup [\![\pi]\!]^\bullet$).

Let us define the *norm* of a configuration C as $\|C\| \stackrel{\text{def}}{=} \max\{C(q) : q \in [\![C]\!]\}$. We obtain:

▶ **Proposition 11.** *Let $\mathcal{P} = (Q, T, I, L, O)$ be a k-way population protocol and let $C_1 \xrightarrow{\pi} C_2$ be a finite execution of $\mathcal{P}$. There exists a finite execution $C_1' \xrightarrow{\pi'} C_2'$ such that (a) $[\![C_1']\!] = [\![C_1]\!]$, (b) $[\![C_2']\!] = [\![C_1]\!] \cup [\![\pi']\!]^\bullet$, and (c) $\|C_1'\| \leq (k+1)^{|Q|}$.*

[3] We could also require the seemingly weaker property that eventually at least one agent "knows". However, by adding transitions that "attract" all other agents to Q_1, we can transform a protocol in which some agent "knows" into a protocol computing the same predicate in which all agents "know".

▶ **Theorem 12.** *Every 1-aware, leaderless and 2-way population protocol* $\mathcal{P} = (Q, T, \{q_0\}, \mathbf{0}, O)$ *computing* $x \geq n$ *has at least* $\log_3 n$ *states.*

Proof. Let $Q_1 \subseteq Q$ be the set of states from the definition of 1-awareness. Since $L = \mathbf{0}$, $C_0 = \wr n \cdot q_0 \wr$ is the smallest initial configuration with output 1, and by Lemma 10 the smallest initial configuration from which some state $q_1 \in Q_1$ is coverable. Let $C_0 \xrightarrow{\pi} C \geq \wr q_1 \wr$. Since $q_1 \neq q_0$, we have $q_1 \in [\![\pi]\!]^\bullet$. By Proposition 11, and since $\mathcal{P}$ is 2-way, q_1 is also coverable from C_0' satisfying $[\![C_0']\!] = [\![C_0]\!] = \{q_0\}$ and $\|C_0'\| = 3^{|Q|}$. Thus, $C_0' = \wr 3^{|Q|} \cdot q_0 \wr$. By minimality of n, we get $n \leq 3^{|Q|}$, and thus $|Q| \geq \log_3 n$. ◀

Observe that the proof Theorem 12 uses the fact that $\mathcal{P}$ is leaderless to conclude $C_0' = \wr 3^{|Q|} \cdot q_0 \wr$ from $[\![C_0']\!] = [\![C_0]\!]$ and $\|C_0'\| = 3^{|Q|}$, which is not necessarily true with leaders.

Protocols with leaders. In the case of protocols with leaders we obtain a lower bound from Rackoff's procedure for the coverability problem of vector addition systems [17].

A *vector addition system* of dimension k (k-VAS) is a pair $(A, \boldsymbol{v}_0)$, where $\boldsymbol{v}_0 \in \mathbb{N}^k$ is an initial vector and $A \subseteq \mathbb{Z}^k$ is a set of vectors. An execution of a k-VAS is a sequence $\boldsymbol{v}_0\boldsymbol{v}_1 \cdots \boldsymbol{v}_n$ of vectors of $\mathbb{N}^k$ such that each $\boldsymbol{v}_{i+1} = \boldsymbol{v}_i + \boldsymbol{a}_i$ for some $\boldsymbol{a}_i \in A$. We write $\boldsymbol{v}_0 \xrightarrow{*} \boldsymbol{v}_n$ and say that the execution has *length* n. A vector $\boldsymbol{v}$ is *coverable* in $(A, \boldsymbol{v}_0)$ if $\boldsymbol{v}_0 \xrightarrow{*} \boldsymbol{v}'$ for some $\boldsymbol{v}' \geq \boldsymbol{v}$. The *size* of a vector $\boldsymbol{v} \in \mathbb{Z}^k$ is $\sum_{1 \leq i \leq k} \text{size}(\max(|\boldsymbol{v}(i)|, 1))$. The *size* of a set of vectors is the sum of the size of its vectors. In [17] Rackoff proves:

▶ **Theorem 13** ([17])**.** *Let* $A \subseteq \mathbb{Z}^k$ *be a set of vectors of size at most* n *and dimension* $k \leq n$*, and let* $\boldsymbol{v}_0 \in \mathbb{N}^k$ *be a vector of size* n*. For every* $\boldsymbol{v} \in \mathbb{N}^k$*, if* $\boldsymbol{v}$ *is coverable in* $(A, \boldsymbol{v}_0)$*, then* $\boldsymbol{v}$ *is coverable by means of an execution of length at most* $2^{(3n)^n}$*.*

Using a standard construction from the Petri net literature, it can be shown that every 2-way protocol $\mathcal{P}$ with n states can be simulated by a VAS $\mathcal{V}_\mathcal{P}$ of size at most $12n^8$, where each execution of $\mathcal{P}$ has a corresponding execution twice as long in $\mathcal{V}_\mathcal{P}$. Thus, by Theorem 13:

▶ **Proposition 14.** *Let* $\mathcal{P} = (Q, T, I, L, O)$ *be a 2-way population protocol and let* $q \in Q$*. For every configuration* C*, if* q *is coverable from* C*, then it is coverable by means of a finite execution of length at most* $2^{(3m)^m - 1}$ *where* $m = 12|Q|^8$*.*

Using the above proposition, we derive:

▶ **Theorem 15.** *Let* $\mathcal{P}$ *be a 1-aware and 2-way population protocol. For every* $n \geq 2$*, if* $\mathcal{P}$ *computes* $x \geq n$*, then* $\mathcal{P}$ *has at least* $(\log\log(n)/151)^{1/9}$ *states.*

6 Protocols for systems of linear inequalities

In Section 3, we have shown that the predicate $x \geq c$ can be computed by a leaderless protocol with $O(\log c)$ states. In this section, we will see that adding a few leaders allows to compute systems of linear inequalities. More formally, we show that there exists a protocol with $O((m+k) \cdot \log(dm))$ states and $O(m \cdot \log(dm))$ leaders computing the predicate $A\boldsymbol{x} \geq \boldsymbol{c}$, where $A \in \mathbb{Z}^{m \times k}$, $\boldsymbol{c} \in \mathbb{Z}^m$ and d is the the largest absolute value occuring in A and $\boldsymbol{c}$.

There are three crucial points that make systems of linear inequalities more complicated than flock-of-birds predicates: (1) variables have *coefficients*, (2) coefficients may be positive or *negative*, and (3) they are the *conjunction* of linear inequalities. We will explain how to address the two first points by considering the special case of linear inequalities. We will then discuss how to handle the third point.

Linear inequalities. Note that the predicate $\sum_{1\leq i\leq k} a_i x_i \geq c$ is equivalent to $\sum_{1\leq i\leq k} a_i x_i + (1-c) > 0$. Therefore, it suffices to describe protocols for predicates of the form $\sum_{1\leq i\leq k} a_i x_i + c > 0$. In order to make the presentation more pleasant, we will first restrain ourselves to the predicate $ax - by + c > 0$ for some fixed $a, b \in \mathbb{N}$ and $c \in \mathbb{Z}$. Such a predicate admits the difficult aspects, i.e. coefficients and negative numbers. Moreover, as we will see, handling more than two variables is not an issue.

Let us now describe a protocol $\mathcal{P}_{\text{lin}}$ for the predicate $ax - by + c > 0$. The idea is to keep a representation of $ax - by + c$ throughout executions of the protocol. Let $n \stackrel{\text{def}}{=} \text{size}(\max(\log|a|, \log|b|, \log|c|, 1))$. As in Section 3, we construct states to represent powers of two. However, this time, we also need states to represent negative numbers:

$$Q^+ \stackrel{\text{def}}{=} \{+\mathbf{2^i} : 0 \leq i \leq n\} \quad \text{and} \quad Q^- \stackrel{\text{def}}{=} \{-\mathbf{2^i} : 0 \leq i \leq n\}.$$

We also need states $X \stackrel{\text{def}}{=} \{\mathbf{x}, \mathbf{y}\}$ for the variables, and two additional states $R \stackrel{\text{def}}{=} \{+\mathbf{0}, -\mathbf{0}\}$. The set of all states of $\mathcal{P}_{\text{lin}}$ is $Q \stackrel{\text{def}}{=} X \cup Q^+ \cup Q^- \cup R$, and the initial states are $I \stackrel{\text{def}}{=} X$.

Let us explain the purpose of R. Intuitively, we would like to have the transitions:

$$x \mapsto \{\!|+\mathbf{2^i} : i \in \text{bits}(a)|\!\} \quad \text{and} \quad y \mapsto \{\!|-\mathbf{2^i} : i \in \text{bits}(|b|)|\!\}.$$

This way, every agent in state $\mathbf{x}$ (resp. $\mathbf{y}$) could be converted to the binary representation of a (resp. b). Unfortunately, this is not possible as these transitions produce more states than they consume. This is where leaders become useful. If R initially contains enough leaders, then R can act as a *reservoir* of extra states which allow to "pad" transitions. More formally, let $\text{rep}(z) \colon \mathbb{Z} \to \text{Pop}(Q \setminus X)$ be defined as follows:

$$\text{rep}(z) \stackrel{\text{def}}{=} \begin{cases} \{\!|+\mathbf{2^i} : i \in \text{bits}(z)|\!\} & \text{if } z > 0, \\ \{\!|-\mathbf{2^i} : i \in \text{bits}(|z|)|\!\} & \text{if } z < 0, \\ \{\!|-\mathbf{0}|\!\} & \text{if } z = 0. \end{cases}$$

For every $r \in R$, we add to $\mathcal{P}_{\text{lin}}$ the following transitions:

$$\text{add}_{\mathbf{x},r} : \mathbf{x}, \underbrace{r, r, \ldots, r}_{|\text{rep}(a)|-1 \text{ times}} \mapsto \text{rep}(a) \quad \text{and} \quad \text{add}_{\mathbf{y},r} : \mathbf{y}, \underbrace{r, r, \ldots, r}_{|\text{rep}(b)|-1 \text{ times}} \mapsto \text{rep}(b).$$

We set the leaders to $L \stackrel{\text{def}}{=} \text{rep}(c) + \{\!|(4n+2) \cdot -\mathbf{0}|\!\}$. We claim that $4n+2$ reservoir states are enough, we will explain later why. Now, the key idea of the construction is that it is always possible to put $2n$ agents back into R. Thus, fairness ensures that the number of agents in X eventually decreases to zero, and then that the value represented over $Q^+ \cup Q^-$ is $ax - by + c$. We let the representations over Q^+ and Q^- "cancel out" until one side "wins". If the positive (resp. negative) side wins, i.e. if $ax - by + c > 0$ (resp. $ax - by + c \leq 0$), then it signals all agents in R to move to $+\mathbf{0}$ (resp. $-\mathbf{0}$). To achieve this, for every $0 \leq i \leq n$, we add transition $\text{cancel}_i : +\mathbf{2^i}, -\mathbf{2^i} \mapsto +\mathbf{0}, -\mathbf{0}$ to the protocol. Since bits of the positive and negative numbers may not be "aligned", we follow the idea of Section 3 and add further transitions to change representations to equivalent ones:

$$\text{up}_i^+ : +\mathbf{2^i}, +\mathbf{2^i} \mapsto +\mathbf{2^{i+1}}, +\mathbf{0}, \qquad \text{down}_{i+1,r}^+ : +\mathbf{2^{i+1}}, r \mapsto +\mathbf{2^i}, +\mathbf{2^i},$$
$$\text{up}_i^- : -\mathbf{2^i}, -\mathbf{2^i} \mapsto -\mathbf{2^{i+1}}, -\mathbf{0}, \qquad \text{down}_{i+1,r}^- : -\mathbf{2^{i+1}}, r \mapsto -\mathbf{2^i}, -\mathbf{2^i},$$

where $0 \leq i < n$ and $r \in R$. Finally, for every $0 \leq i \leq n$, we add transitions to signal which side wins:

$$\text{signal}_i^+ : +\mathbf{2^i}, -\mathbf{0} \mapsto +\mathbf{2^i}, +\mathbf{0}, \qquad \text{signal} : -\mathbf{0}, +\mathbf{0} \mapsto -\mathbf{0}, -\mathbf{0},$$
$$\text{signal}_i^- : -\mathbf{2^i}, +\mathbf{0} \mapsto -\mathbf{2^i}, -\mathbf{0}.$$

Note that $\mathbf{-0}$ "wins" over $\mathbf{+0}$ because the predicate is false whenever $ax - by + c = 0$. It remains to specify the output mapping of $\mathcal{P}_{\text{lin}}$ which we define as expected, i.e. $O(q) \stackrel{\text{def}}{=} 1$ if $q \in Q^+ \cup \{\mathbf{+0}\}$, and $O(q) \stackrel{\text{def}}{=} 0$ otherwise.

Let us briefly explain why $4n + 2$ reservoir states suffice. At any reachable configuration C, transitions of the form up_i^+ and up_i^- can occur until $C(\pm\mathbf{2^i}) \leq 1$ for every $0 \leq i < n$. Afterwards, at most $2n$ agents remain in these states. There can however be many agents in $S = \{\mathbf{+2^n}, \mathbf{-2^n}\}$. But, these two states represent numbers respectively larger and smaller than any coefficient, hence the number of agents in S can only grow by one each time a state from X is consumed. Overall, this means that $C \xrightarrow{*} C'$ for some C' such that $C'(R) \geq 2n$.

In order to handle more variables $\{x_1, x_2, \ldots, x_k\}$, note that all we need to do is to set $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k}\}$ instead, and add transitions $\text{add}_{\mathbf{x_i},r}$ for every $1 \leq i \leq k$ and $r \in R$.

By applying Lemma 3 on $\mathcal{P}_{\text{lin}}$, we obtain:

▶ **Theorem 16.** *Let $a_1, a_2, \ldots, a_k, c \in \mathbb{Z}$ and let $n = \text{size}(\max(|a_1|, |a_2|, \ldots, |a_k|, |c|, 1))$. There exists a 2-way population protocol, with at most $10kn$ states and at most $5n + 2$ leaders, that computes the predicate $\sum_{1 \leq i \leq k} a_i x_i + c > 0$.*

Conjunction of linear inequalities. We briefly explain how to lift the construction for linear inequalities to systems of linear inequalities. The details of the formal construction and proofs are a bit involved, and are thus deferred to the full version of this paper. Let us fix some $A \in \mathbb{Z}^{m \times k}$ and $\boldsymbol{c} \in \mathbb{Z}^m$. We sketch a protocol $\mathcal{P}_{\text{sys}}$ for the predicate $A\boldsymbol{x} + \boldsymbol{c} > \mathbf{0}$. For every $1 \leq i \leq m$, we construct a protocol $\mathcal{P}_i$ for the predicate $\sum_{1 \leq j \leq k} A_{i,j} \cdot \boldsymbol{x}_j + \boldsymbol{c}_i > 0$. Protocol $\mathcal{P}_i$ is obtained as presented earlier, but with some modifications. The largest power of two is picked as $n \stackrel{\text{def}}{=} \text{size}(d) + \lceil \log 2m^2 \rceil$ where

$$d \stackrel{\text{def}}{=} \max(1, \{|A_{i,j}| : 1 \leq i \leq m, 1 \leq j \leq k\}, \{|c_i| : 1 \leq i \leq m\}).$$

The reason for this modification is that the number of agents, in a largest power of two, should now increase by at most $1/m$ each time an initial state is consumed, as opposed to 1.

We also replace each *positive state* $q \in Q^+$ of $\mathcal{P}_i$ by two states q_0 and q_1, its *0-copy* and *1-copy*. The reason behind this is that positive states should not necessarily have output 1. Indeed, one linear inequality may be satisfied while the other ones are not. Therefore, $\mathbf{-0}$ and each *negative state* $q \in Q^-$ should be able to signal a 0-consensus to the positive states. The transitions of the form up_j^+, down_j^+ and cancel_j are adapted accordingly.

Protocol $\mathcal{P}_{\text{sys}}$ is obtained as follows. First, subprotocols $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m$ are put side by side. Their initial (resp. reservoir) states are merged into a single set X (resp. R). For every $1 \leq j \leq k$, transitions $\text{add}_{\mathbf{x_j},r}$ of the m subprotocols are replaced by a single transition consuming $\mathbf{x_j}$, and enough reservoir states, and producing $\text{rep}(A_{i,j})$ in each subprotocol $\mathcal{P}_i$, where $1 \leq i \leq m$. The signal mechanisms are replaced by these new ones:

- the 0-copy of state $\mathbf{+2^0}$ of *all* subprotocols can meet to convert $\mathbf{-0}$ to $\mathbf{+0}$,
- state $\mathbf{+0}$ can convert any positive state to its 1-copy,
- state $\mathbf{-0}$ or any negative state can convert $\mathbf{+0}$ to $\mathbf{-0}$, and any positive state to its 0-copy.

A careful analysis of the formal construction of $\mathcal{P}_{\text{sys}}$ combined with Lemma 3 yields:

▶ **Theorem 17.** *Let $A \in \mathbb{Z}^{m \times k}$, $\boldsymbol{c} \in \mathbb{Z}^m$ and $n = \text{size}(\max(1, \{|A_{i,j}| : 1 \leq i \leq m, 1 \leq j \leq k\}), \{|c_i| : 1 \leq i \leq m\})$. There exists a 2-way population protocol, with at most $27(\log m + n)(m + k)$ states and at most $14m(\log m + n)$ leaders, that computes the predicate $A\boldsymbol{x} + \boldsymbol{c} > \mathbf{0}$.*

7 Conclusion and further work

We have initiated the study of the state space size of population protocols as a function of the size of the predicate they compute. Previous lower bounds were only for single predicates, like the majority predicate $x \leq y$, or for a variant of the model in which the number of states is a function of the number of agents.

There are many open questions. We conjecture that systems of linear inequalities can be computed by leaderless protocols with a polynomial number of states. A second, very intriguing question is whether the function $f(n)$ giving the minimal number of states of a two-leader protocol computing $x \geq n$ exhibits large gaps, i.e., if there are (families of) numbers c and $c+1$ such that $f(c)$ is exponentially larger than $f(c+1)$. A third question is whether there exist protocols with $O(\log\log\log n)$ states for the flock-of-birds predicates $x \geq n$. Such protocols cannot be 1-aware, but they might exist. Their existence is linked to the long standing question of whether the reachability problem for reversible VAS (a model equivalent to the commutative semigroup representations of [15]) has the same complexity as reachability for arbitrary VAS (see [12] for a brief introduction).

References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in population protocols. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2560–2579. SIAM, 2017. doi:10.1137/1.9781611974782.169.

2 Dan Alistarh, Rati Gelashvili, and Milan Vojnovic. Fast and exact majority in population protocols. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 47–56. ACM, 2015. doi:10.1145/2767386.2767429.

3 Dana Angluin, James Aspnes, Melody Chan, Michael J. Fischer, Hong Jiang, and René Peralta. Stably computable properties of network graphs. In Viktor K. Prasanna, S. Sitharama Iyengar, Paul G. Spirakis, and Matt Welsh, editors, *Distributed Computing in Sensor Systems, First IEEE International Conference, DCOSS 2005, Marina del Rey, CA, USA, June 30 - July 1, 2005, Proceedings*, volume 3560 of *Lecture Notes in Computer Science*, pages 63–74. Springer, 2005. doi:10.1007/11502593_8.

4 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In Soma Chaudhuri and Shay Kutten, editors, *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004, St. John's, Newfoundland, Canada, July 25-28, 2004*, pages 290–299. ACM, 2004. doi:10.1145/1011767.1011810.

5 Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(3):183–199, 2008. doi:10.1007/s00446-008-0067-z.

6 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007. doi:10.1007/s00446-007-0040-2.

7 James Aspnes. Clocked population protocols. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 431–440. ACM, 2017. doi:10.1145/3087801.3087836.

8 James Aspnes and Eric Ruppert. An introduction to population protocols. In *Middleware for Network Eccentric and Mobile Applications*, pages 97–120. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-540-89707-1_5.

9 Andreas Bilke, Colin Cooper, Robert Elsässer, and Tomasz Radzik. Brief announcement: Population protocols for leader election and exact majority with $O(\log^2 n)$ states and $O(\log^2 n)$ convergence time. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 451–453. ACM, 2017. doi:10.1145/3087801.3087858.

10 David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. In Yoram Moses, editor, *Distributed Computing - 29th International Symposium, DISC 2015, Tokyo, Japan, October 7-9, 2015, Proceedings*, volume 9363 of *Lecture Notes in Computer Science*, pages 602–616. Springer, 2015. doi:10.1007/978-3-662-48653-5_40.

11 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Inf.*, 54(2):191–215, 2017. doi:10.1007/s00236-016-0272-3.

12 Alain Finkel and Jérôme Leroux. Recent and simple algorithms for petri nets. *Software and System Modeling*, 14(2):719–725, 2015. doi:10.1007/s10270-014-0426-0.

13 Rachid Guerraoui and Eric Ruppert. Names trump malice: Tiny mobile agents can tolerate byzantine failures. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th Internatilonal Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 484–495. Springer, 2009. doi:10.1007/978-3-642-02930-1_40.

14 Giuseppe Antonio Di Luna, Paola Flocchini, Taisuke Izumi, Tomoko Izumi, Nicola Santoro, and Giovanni Viglietta. Population protocols with faulty interactions: The impact of a leader. In Dimitris Fotakis, Aris Pagourtzis, and Vangelis Th. Paschos, editors, *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, volume 10236 of *Lecture Notes in Computer Science*, pages 454–466, 2017. doi:10.1007/978-3-319-57586-5_38.

15 Ernst W. Mayr and Albert R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46(3):305–329, 1982. doi:10.1016/0001-8708(82)90048-2.

16 Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Mediated population protocols. *Theor. Comput. Sci.*, 412(22):2434–2450, 2011. doi:10.1016/j.tcs.2011.02.003.

17 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

18 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7(4):615–633, 2008. doi:10.1007/s11047-008-9067-y.

Communicating Finite-State Machines and Two-Variable Logic

Benedikt Bollig
LSV, CNRS & ENS Paris-Saclay, Université Paris-Saclay, France

Marie Fortin
LSV, CNRS & ENS Paris-Saclay, Université Paris-Saclay, France

Paul Gastin
LSV, CNRS & ENS Paris-Saclay, Université Paris-Saclay, France

Abstract

Communicating finite-state machines are a fundamental, well-studied model of finite-state processes that communicate via unbounded first-in first-out channels. We show that they are expressively equivalent to existential MSO logic with two first-order variables and the order relation.

2012 ACM Subject Classification Theory of computation → Concurrency, Theory of computation → Logic and verification

Keywords and phrases Communicating Finite-state Machines, MSO logic, Message Sequence Charts

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.17

Related Version A full version of the paper is available at `https://arxiv.org/abs/1709.09991`.

1 Introduction

The study of logic-automata connections has ever played a key role in computer science, relating concepts that are a priori very different. Its motivation is at least twofold. First, automata may serve as a tool to decide logical theories. Beginning with the work of Büchi, Elgot, and Trakhtenbrot, who established expressive equivalence of monadic second-order (MSO) logic and finite automata [8, 9, 26], the "automata-theoretic" approach to logic has been successfully applied, for example, to MSO logic on trees [23], temporal logics [27], and first-order logic with two variables over words with an equivalence relation (aka *data words*) [4]. Second, automata serve as models of various kind of state-based systems. Against this background, Büchi-like theorems lay the foundation of *synthesis*, i.e., the process of transforming high-level specifications (represented as logic formulas) into faithful system models. In this paper, we provide a Büchi theorem for *communicating finite-state machines*, which are a classical model of concurrent message-passing systems.

One of the simplest system models are finite automata. They can be considered as single finite-state processes and, therefore, serve as a model of *sequential* systems. Their executions are words, which, seen as a logical structure, consist of a set of positions (also referred to as *events*) that carry letters from a finite alphabet and are *linearly* ordered by some binary relation $\leq$. The simple MSO (even first-order) formula $\forall x.\big(a(x) \implies \exists y.(x \leq y \wedge b(y))\big)$ says that every "request" a is eventually followed by an "acknowledgment" b. In fact, Büchi's theorem allows one to turn any logical MSO specification into a finite automaton. The latter

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 17; pp. 17:1–17:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

can then be considered correct by construction. Though the situation quickly becomes more intricate when we turn to other automata models, Büchi theorems have been established for expressive generalizations of finite automata that also constitute natural system models. In the following, we will discuss some of them.

Data automata accept (in the context of system models, we may also say *generate*) words that, in addition to the linear order $\leq$ and its direct-successor relation, are equipped with an equivalence relation $\sim$ [4]. Positions (events) that belong to the same equivalence class may be considered as being executed by one and the same process, while $\leq$ reflects a sort of global control. It is, therefore, convenient to also include a predicate that connects *successive* events in an equivalence class. Bojańczyk et al. showed that data automata are expressively equivalent to existential MSO logic with two first-order variables [4]. A typical formula is $\neg\exists x.\exists y.(x \neq y \wedge x \sim y)$, which says that every equivalence class is a singleton. It should be noted that data automata scan a word twice and, therefore, can hardly be seen as a system model. However, they are expressively equivalent to *class-memory automata*, which distinguish between a global control (modeling, e.g., a shared variable) and a local control for every process [3].

Unlike finite automata and data automata, *asynchronous automata* are a model of *concurrent* shared-memory systems, with a finite number of processes. Their executions are Mazurkiewicz traces, where the relation $\leq$ is no longer a total, but a partial order. Thus, there may be *parallel* events x and y, for which neither $x \leq y$ nor $y \leq x$ holds. A typical logical specification is the mutual exclusion property, which can be expressed in MSO logic as $\neg\exists x.\exists y.(CS(x) \wedge CS(y) \wedge x \parallel y)$ where the parallel operator $x \parallel y$ is defined as $\neg(x \leq y) \wedge \neg(y \leq x)$. Note that this is even a first-order formula that uses only two first-order variables, x and y. It says that there are no two events x and y that access a critical section simultaneously. Asynchronous automata are closed under complementation [29] so that the inductive approach to translating formulas into automata can be applied to obtain a Büchi theorem [24]. Note that complementability is also the key ingredient for MSO characterizations of *nested-word automata* [1] and *branching automata* running over series-parallel posets (aka N-free posets) [18, 2].

The situation is quite different in the realm of *communicating finite-state machines (CFMs)*, aka *communicating automata* or *message-passing automata*, where finitely many processes communicate by exchanging messages through *unbounded* FIFO channels [7]. A CFM accepts/generates message-sequence charts (MSCs) which are also equipped with a partial order $\leq$. Additional binary predicates connect (i) the emission of a message with its reception, and (ii) successive events executed by one and the same process. Unfortunately, CFMs are not closed under complementation [6] so that an inductive translation of MSO logic into automata will fail. In fact, they are strictly less expressive than MSO logic. Two approaches have been adopted to overcome these problems. First, when channels are (existentially or universally) bounded, closure under complementation is recovered so that CFMs are expressively equivalent to MSO logic [17, 19, 11, 12]. Note that, however, the corresponding proofs are much more intricate than in the case of finite automata. Second, CFMs with unbounded channels have been shown to be expressively equivalent to *existential* MSO logic when dropping the order $\leq$ [6]. The proof relies on Hanf's normal form of first-order formulas on structures of *bounded degree* (which is why one has to discard $\leq$) [15]. However, it is clear that many specifications (such as mutual exclusion) are easier to express in terms of $\leq$. But, to the best of our knowledge, a convenient specification language that is exactly as expressive as CFMs has still been missing.

It is the aim of this paper to close this gap, i.e., to provide a logic that

- *matches* exactly the expressive power of *unrestricted* CFMs (in particular, every specification should be *realizable* as an automaton), and
- *includes* the order $\leq$ so that one can easily express natural properties like mutual exclusion.

We show that existential MSO logic with two first-order variables is an appropriate logic. To translate a formula into an automaton, we first follow the approach of [4] for data automata and consider its Scott normal form (cf. [13]). However, while data automata generate total orders, the main difficulty in our proof comes from the fact that $\leq$ is a partial order. Actually, our main technical contribution is a CFM that, running on an MSC, marks precisely those events that are in parallel to some event of a certain type.

It should be noted that message-passing automata can also be used as acceptors of the underlying (graph) architecture. In that case, logical characterizations have been obtained in terms of MSO and modal logics [20, 16, 21, 22]. However, in our framework, the architecture is fixed and we rather reason about the set of *executions* of a CFM.

The paper is structured as follows. In Section 2, we recall the classical notions of CFMs and MSO logic. Section 3 states our main result, describes our proof strategy, and settles several preliminary lemmas. The main technical part is contained in Section 4. We conclude in Section 5. Missing proofs can be found at: `https://arxiv.org/abs/1709.09991`.

2 Preliminaries

Let Σ be a finite alphabet. The set of finite words over Σ is denoted by Σ^*, which includes the empty word ε. For $w \in \Sigma^*$, let $|w|$ denote its length. In particular, $|\varepsilon| = 0$. The inverse of a binary relation R is defined as $R^{-1} = \{(f, e) \mid (e, f) \in R\}$. We denote the size of a finite set A by $|A|$. The disjoint union of sets A and B is denoted by $A \uplus B$.

2.1 Communicating Finite-State Machines

Communicating finite-state machines are a natural model of communicating systems where a finite number of processes communicate through a priori unbounded FIFO channels [7]. Every process is represented as a *finite transition system* (S, ι, Δ) over some finite alphabet Γ, i.e., S is a finite set of states with initial state $\iota \in S$, and $\Delta \subseteq S \times \Gamma \times S$ is the transition relation. An element from Γ will describe the action that is performed when taking a transition (e.g., "send a message to some process" or "perform a local computation").

A communicating finite-state machine is a *collection* of finite transition systems, one for each process. We assume a finite set $P = \{p, q, r, \ldots\}$ of *processes* and a finite alphabet $\Sigma = \{a, b, c, \ldots\}$ of *labels*. We suppose that there is a channel between any two distinct processes. Thus, the set of *channels* is $Ch = \{(p, q) \in P \times P \mid p \neq q\}$.

▶ **Definition 1.** A *communicating finite-state machine (CFM)* over P and Σ is a tuple $\mathcal{A} = ((\mathcal{A}_p)_{p \in P}, Msg, Acc)$ where

- Msg is a finite set of *messages*,
- $\mathcal{A}_p = (S_p, \iota_p, \Delta_p)$ is a finite transition system over $\Sigma \cup (\Sigma \times \{!, ?\} \times Msg \times (P \setminus \{p\}))$, and
- $Acc \subseteq \prod_{p \in P} S_p$ is the set of *global accepting states*.[1]

[1] We may also include several *global* initial states without changing the expressive power, which is convenient in several of the forthcoming constructions.

Let $t = (s, \alpha, s') \in \Delta_p$ be a transition of process p. We call s the *source state* of t, denoted by $source(t)$, and s' its *target state*, denoted $target(t)$. Moreover, α is the *action* executed by t. If $\alpha \in \Sigma$, then t is said to be *internal*, and we let $label(t) = \alpha$. The label from Σ may provide some more information about an event (such as "enter critical section"). When α is of the form $(a, !, \mathrm{m}, q)$, then t is a send transition, which writes message m into the channel (p, q). Accordingly, we let $msg(t) = \mathrm{m}$, $receiver(t) = q$, and $label(t) = a$. Finally, performing $\alpha = (a, ?, \mathrm{m}, q)$ removes message m from channel (q, p). In that case, we set $msg(t) = \mathrm{m}$, $sender(t) = q$, and $label(t) = a$.

If there is only one process, i.e., P is a singleton, then all transitions are internal so that a CFM is simply a finite automaton accepting a regular set of words over the alphabet Σ. In the presence of several processes, a single behavior is a *tuple* of words over Σ, one for every process. However, these words are not completely independent (unless all transitions are internal and there is no communication), since the sending of a message can be linked to its reception. This is naturally reflected by a binary relation $\lhd$ that connects word positions on distinct processes. The resulting structure is called a message sequence chart.

▶ **Definition 2.** A *message sequence chart (MSC)* over P and Σ is a tuple $M = ((w_p)_{p\in P}, \lhd)$ where $w_p \in \Sigma^*$ for every $p \in P$. We require that at least one of these words be non-empty. By $E_p = \{p\} \times \{1, \dots, |w_p|\}$, we denote the set of *events* that are executed by process p. Accordingly, the (disjoint) union $E = \bigcup_{p\in P} E_p$ is the set of all events. Implicitly, we obtain the labeling $\lambda : E \to (P \times \Sigma)$ defined by $\lambda((p, i)) = (p, a)$ where a is the i-th letter of w_p, and the process-edge relation $\to \subseteq \bigcup_{p\in P}(E_p \times E_p)$, which connects successive events that are executed by one and the same process: $(p, i) \to (p, i+1)$ for all $p \in P$ and $i \in \{1, \dots, |w_p|-1\}$. Now, $\lhd \subseteq \bigcup_{(p,q)\in Ch}(E_p \times E_q)$ is a set of *message edges*, satisfying the following:

- $(\to \cup \lhd)$ is acyclic (intuitively, messages cannot travel backwards in time), and the associated partial order is denoted $\leq = (\to \cup \lhd)^*$ with strict part $< = (\to \cup \lhd)^+$,
- each event is part of at most one message edge, and
- for all $(p, q) \in Ch$ and $(e, f), (e', f') \in \lhd \cap (E_p \times E_q)$, we have $e \to^* e'$ iff $f \to^* f'$ (which guarantees a FIFO behavior).

An event that does not belong to a message edge is called *internal*. We say that two events $e, f \in E$ are *parallel*, written $e \parallel f$, if neither $e \leq f$ nor $f \leq e$. The set of all MSCs is denoted $\mathbb{MSC}(P, \Sigma)$.

▶ **Example 3.** An example MSC over $P = \{p, q, r\}$ and $\Sigma = \{a, b, c\}$ is depicted in Figure 1. For the moment, we ignore the colored areas and annotations. We have $w_p = aacaaaaa$, $w_r = aaaaaaaaaa$, and $w_q = abbaacaaa$ (note that q is the bottom process). Consider the events $f = (p, 4)$, $e = (p, 5)$, and $g = (q, 2)$. That is, $f \to e$ and $g \lhd e$. Moreover, $(p, 3) \parallel (q, 6)$ (i.e., the two c-labeled events are parallel), while $(p, 3) \leq (q, 8)$.

Let $M = ((w_p)_{p\in P}, \lhd)$ be an MSC over P and Σ. A *run* of the CFM $\mathcal{A}$ on M is given by a mapping ρ that associates with every event $e \in E_p$ ($p \in P$) the transition $\rho(e) \in \Delta_p$ that is executed at e. We require that

1. for every $e \in E$ with $\lambda(e) = (p, a)$, we have $label(\rho(e)) = a$,
2. for every process $p \in P$ such that $E_p \neq \emptyset$, we have $source(\rho((p, 1))) = \iota_p$,
3. for every process edge $(e, f) \in \to$, we have $target(\rho(e)) = source(\rho(f))$,
4. for every internal event $e \in E$, $\rho(e)$ is an internal transition, and
5. for every message edge $(e, f) \in \lhd$ with $e \in E_p$ and $f \in E_q$, $\rho(e) \in \Delta_p$ is a send transition and $\rho(f) \in \Delta_q$ is a receive transition such that $msg(\rho(e)) = msg(\rho(f))$, $receiver(\rho(e)) = q$, and $sender(\rho(f)) = p$.

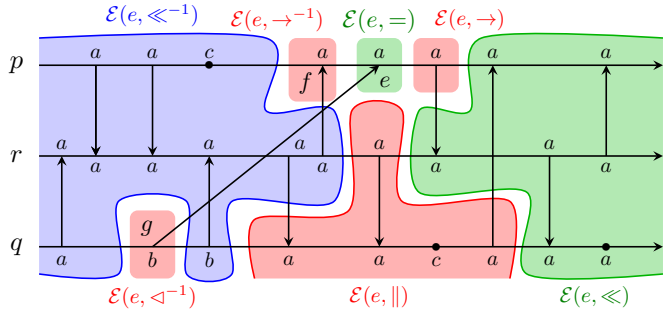


Figure 1 An MSC (cf. Example 3) and the partition determined by an event e (cf. Example 10).

Note that, when $|P| = 1$, Condition 5. becomes meaningless and Conditions 1.–4. emulate the behavior of a finite automaton.

It remains to define when ρ is accepting. To this aim, we collect the final states of each process p. If $E_p \neq \emptyset$, then let s_p be the target state of $\rho((p, |w_p|))$, i.e., of the last transition taken by p. Otherwise, let $s_p = \iota_p$. Now, we say that ρ is *accepting* if $(s_p)_{p \in P} \in Acc$.

Finally, the *language* of $\mathcal{A}$ is defined as $L(\mathcal{A}) = \{M \in \mathbb{MSC}(P, \Sigma) \mid \text{there is an accepting run of } \mathcal{A} \text{ on } M\}$.

2.2 MSO and Two-Variable Logic

While CFMs serve as an operational model of concurrent systems, MSO logic can be considered as a high-level specification language. It uses first-order variables $x, y, \ldots$ to quantify over events, and second-order variables $X, Y, \ldots$ to represent sets of events. The logic MSO (we assume that P and Σ are understood from the context) is defined by the following grammar:

$$\varphi ::= p(x) \mid a(x) \mid x \in X \mid x = y \mid x \to y \mid x \lhd y \mid x \leq y \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists x.\varphi \mid \exists X.\varphi$$

where x and y are first-order variables, X is a second-order variable, $p \in P$, and $a \in \Sigma$. We use the usual operator precedence. For convenience, we allow usual abbreviations such as conjunction $\varphi \wedge \psi$, universal quantification $\forall x.\varphi$, implication $\varphi \implies \psi$, etc. The atomic formulas $p(x)$ and $a(x)$ are interpreted as "x is located on process p" and, respectively, "the label of event x is a". The binary predicates are self-explanatory, and the boolean connectives and quantification are interpreted as usual. The size $|\varphi|$ of a formula $\varphi \in$ MSO is the length of φ seen as a string.

A variable that occurs free in a formula requires an interpretation in terms of an event/a set of events from the given MSC. We will write, for example, $M, x \mapsto e, y \mapsto f \models \varphi$ if M satisfies φ provided x is interpreted as e and y as f. If φ is a *sentence* (i.e., does not contain any free variable), then we write $M \models \varphi$ to denote that M satisfies φ. With a sentence φ, we associate the MSC language $L(\varphi) = \{M \in \mathbb{MSC}(P, \Sigma) \mid M \models \varphi\}$.

The set FO of *first-order formulas* is the fragment of MSO that does not make use of second-order quantification $\exists X$. The two-variable fragment of FO, denoted by FO^2, allows only for two first-order variables, x and y (which, however, can be quantified and reused arbitrarily often). Moreover, formulas from EMSO, the *existential fragment* of MSO, are of the form $\exists X_1 \ldots \exists X_n.\varphi$ where $\varphi \in$ FO. Accordingly, EMSO^2 is the set of EMSO formulas whose first-order kernel is in FO^2.

The expressive power of all these fragments heavily depends on the set of binary predicates among $\{\rightarrow, \lhd, \leq\}$ that are actually allowed. For a logic $\mathcal{C} \in \{\mathrm{MSO}, \mathrm{EMSO}, \mathrm{EMSO}^2, \mathrm{FO}, \mathrm{FO}^2\}$ and a set $R \subseteq \{\rightarrow, \lhd, \leq\}$, let $\mathcal{C}[R]$ be the logic $\mathcal{C}$ restricted to the binary predicates from R (however, we always allow for equality, i.e., formulas of the form $x = y$). In particular, $\mathrm{MSO} = \mathrm{MSO}[\rightarrow, \lhd, \leq]$. As the transitive closure of a binary relation is definable in terms of second-order quantification, $\mathrm{MSO}[\rightarrow, \lhd, \leq]$ and $\mathrm{MSO}[\rightarrow, \lhd]$ have the same expressive power (over MSCs). On the other hand, $\mathrm{MSO}[\leq]$ is strictly less expressive [6].

▶ **Example 4.** Suppose $P = \{p, q, r\}$ and $\Sigma = \{a, b, c\}$. The (mutual exclusion) formula $\neg\exists x.\exists y.(c(x) \wedge c(y) \wedge x \parallel y)$, where $x \parallel y$ is defined as $\neg(x \leq y) \wedge \neg(y \leq x)$, is in $\mathrm{FO}^2[\leq]$. It is not satisfied by the MSC from Figure 1, as the two c-labeled internal events are parallel.

Let us turn to the relative expressive power of CFMs and logic. We say that CFMs and a logic $\mathcal{C}$ are *expressively equivalent* if,

- for every CFM $\mathcal{A}$, there exists a sentence $\varphi \in \mathcal{C}$ such that $L(\mathcal{A}) = L(\varphi)$, and
- for every sentence $\varphi \in \mathcal{C}$, there exists a CFM $\mathcal{A}$ such that $L(\mathcal{A}) = L(\varphi)$.

Now, the Büchi-Elgot-Trakhtenbrot theorem can be stated as follows:

▶ **Theorem 5** ([8, 9, 26])**.** *If $|P| = 1$, then CFMs (i.e., finite automata) and* MSO *are expressively equivalent.*

Unfortunately, when several processes are involved, MSO is too expressive to be captured by CFMs, unless one restricts the logic:

▶ **Theorem 6** ([6])**.** *CFMs and* $\mathrm{EMSO}[\rightarrow, \lhd]$ *are expressively equivalent.*

3 Two-Variable Logic and CFMs

The logic $\mathrm{EMSO}[\rightarrow, \lhd]$ is not very convenient as a specification language, as it does not allows us to talk, explicitly, about the order of an MSC. It should be noted that CFMs and MSO are expressively equivalent if one restricts to MSCs that are *channel-bounded* [17, 11, 19]. Our main result allows one to include $\leq$ in the unbounded case, too, though we have to restrict to two first-order variables:

▶ **Theorem 7.** *CFMs and* $\mathrm{EMSO}^2[\rightarrow, \lhd, \leq]$ *are expressively equivalent. More precisely:*

1. *Given a CFM $\mathcal{A}$, we can effectively construct a sentence $\varphi_{\mathcal{A}} \in \mathrm{EMSO}^2[\rightarrow, \lhd, \leq]$ of polynomial size such that $L(\mathcal{A}) = L(\varphi_{\mathcal{A}})$.*
2. *Given a sentence $\varphi \in \mathrm{EMSO}^2[\rightarrow, \lhd, \leq]$, we can effectively construct a CFM $\mathcal{A}_\varphi$ with $2^{2^{\mathcal{O}(|\varphi| + |P| \log |P|)}}$ states (per process) such that $L(\mathcal{A}_\varphi) = L(\varphi)$.*

Translating a CFM into an EMSO^2 formula is standard: Second-order variables represent an assignment of transitions to events. The first-order kernel then checks whether this guess is consistent with the definition of an accepting run.

Before dwelling on the involved proof of the converse translation, i.e., Theorem 7(2), we start with some comments. The CFM $\mathcal{A}_\varphi$ is inherently *nondeterministic* (for the definition of a deterministic CFM, cf. [12]). Already for FO^2, this is unavoidable: CFMs are in general not determinizable, as witnessed by an FO^2-definable language in [12, Proposition 5.1]. Note that the number of states of $\mathcal{A}_\varphi$ is, in fact, independent of the number of letters from Σ that do *not* occur in the formula. This is why Theorem 7(2) mentions only $|\varphi|$ rather than $|\Sigma|$. Actually, the doubly exponential size of $\mathcal{A}_\varphi$ is necessary, even for $\mathrm{FO}^2[\rightarrow]$ or $\mathrm{FO}^2[\leq]$ sentences and a small number of processes. The following can be shown using known techniques [14, 28]:

▶ **Theorem 8.** *(i) Assume $|P| = 1$ and $|\Sigma| = 2$. For all $n \in \mathbb{N}$, there is a sentence $\varphi \in \mathrm{FO}^2[\rightarrow]$ of size $\mathcal{O}(n^2)$ such that no CFM with less than 2^{2^n} states recognizes $L(\varphi)$.*
(ii) Assume $|P| = 2$ and $|\Sigma| = n$ with $n \geq 2$. There is a sentence $\varphi \in \mathrm{FO}^2[\leq]$ of size $\mathcal{O}(n)$ such that no CFM with less than $2^{2^{n-1}}$ states on every process recognizes $L(\varphi)$.

The rest of this paper is devoted to the proof of Theorem 7(2). In a first step, we translate the given formula into Scott normal form (cf. [13]):

▶ **Lemma 9** (Scott Normal Form). *Every sentence from $\mathrm{EMSO}^2[\rightarrow, \lhd, \leq]$ is effectively equivalent to a linear-size sentence of the form $\exists X_1 \ldots \exists X_m.\psi$ where $\psi = \forall x.\forall y.\varphi \wedge \bigwedge_{i=1}^{\ell} \forall x.\exists y.\varphi_i \in \mathrm{FO}^2[\rightarrow, \lhd, \leq]$ with $\varphi, \varphi_1, \ldots, \varphi_\ell$ quantifier-free.*

As the class of languages accepted by CFMs is closed under projection, it remains to deal with the first-order part ψ. Note that ψ contains free occurrences of second-order variables $X_1, \ldots, X_m$. To account for an interpretation of these variables, we extend the alphabet Σ towards the alphabet $\Sigma' = \Sigma \times \{0,1\}^m$ of exponential size. When an event e is labeled with $(a, b_1, \ldots, b_m) \in \Sigma'$, we consider that $e \in X_i$ iff $b_i = 1$.

As the class of languages accepted by CFMs is closed under intersection, too, the proof of Theorem 7(2) comes down to the translation of the formulas of the form $\forall x.\forall y.\eta$ or $\forall x.\exists y.\eta$ where η is a quantifier-free formula with free variables among $X_1, \ldots, X_m, x, y$. Notice that, given an MSC $M \in \mathbb{MSC}(P, \Sigma')$ and events e and f in M, whether $M, x \mapsto e, y \mapsto f \models \eta$ holds or not only depends on the labels of e and f, and their relative position. This is formalized below in terms of *types*.

Types. Let $M = ((w_p)_{p \in P}, \lhd) \in \mathbb{MSC}(P, \Sigma')$ be an MSC. Towards the definition of the type of an event, we define another binary relation $\ll \;=\; < \setminus (\rightarrow \cup \lhd)$. Let Ω be the set of *relation symbols* $\{=, \rightarrow, \lhd, \parallel, \rightarrow^{-1}, \lhd^{-1}, \ll, \ll^{-1}\}$. Given an event $e \in E$ and a relation symbol $\bowtie \in \Omega$, we let $\mathcal{E}(e, \bowtie) = \{f \in E \mid e \bowtie f\}$. In particular, $\mathcal{E}(e, \ll^{-1}) = \{f \in E \mid f < e \wedge \neg(f \rightarrow e) \wedge \neg(f \lhd e)\}$. Notice that all these sets form a partition of E, i.e., $E = \biguplus_{\bowtie \in \Omega} \mathcal{E}(e, \bowtie)$ (some sets may be empty, though). The $\bowtie$-*type* and the *type* of an event $e \in E$ are respectively defined by

$$\mathsf{type}^{\bowtie}_M(e) = \{\lambda(f) \mid f \in \mathcal{E}(e, \bowtie)\} \subseteq P \times \Sigma' \quad \text{and} \quad \mathsf{type}_M(e) = \big(\mathsf{type}^{\bowtie}_M(e)\big)_{\bowtie \in \Omega}.$$

By $\mathbb{T}_{P,\Sigma'} = \prod_{\bowtie \in \Omega} 2^{P \times \Sigma'}$, we denote the (finite) set of possible types. Thus, we actually deal with functions $\mathsf{type}^{\bowtie}_M : E \to 2^{P \times \Sigma'}$ and $\mathsf{type}_M : E \to \mathbb{T}_{P,\Sigma'}$.

▶ **Example 10.** Consider Figure 1 and the distinguished event e. Suppose $a, b, c \in \Sigma'$. The sets $\mathcal{E}(e, \bowtie)$, which form a partition of the set of events, are indicated by the colored areas. Note that, since e is a receive event, $\mathcal{E}(e, \lhd) = \emptyset$. Moreover, $\mathsf{type}^{\rightarrow}_M(e) = \mathsf{type}^{=}_M(e) = \mathsf{type}^{\rightarrow^{-1}}_M(e) = \{(p, a)\}$ and $\mathsf{type}^{\ll^{-1}}_M(e) = \{(p, a), (p, c), (r, a), (q, a), (q, b)\}$.

In fact, it is enough to know the type of every event to (effectively) evaluate ψ. To formalize this, let η be a quantifier-free formula with free variables among $X_1, \ldots, X_m, x, y$. Assume that we are given $M \in \mathbb{MSC}(P, \Sigma')$ and two events e and f that are labeled with $(p, \sigma), (p', \sigma') \in P \times \Sigma'$, respectively, where $\sigma = (a, b_1, \ldots, b_m)$ and $\sigma' = (a', b'_1, \ldots, b'_m)$. Let $\bowtie \in \Omega$ be the unique relation such that $e \bowtie f$. To decide whether $M, x \mapsto e, y \mapsto f \models \eta$, we rewrite η into a propositional formula $[\![\eta]\!]^{\bowtie}_{(p,\sigma),(p',\sigma')}$ that can be evaluated to *true* or *false*: Replace the formulas $p(x)$, $a(x)$, $p'(y)$, $a'(y)$, $x \in X_i$ with $b_i = 1$, and $y \in X_i$ with $b'_i = 1$ by *true*. All other unary predicates become *false* (we consider $z \in X_i$ to be unary). Formulas $z \sqsubset z'$ with $z, z' \in \{x, y\}$ and $\sqsubset \in \{=, \rightarrow, \lhd, \leq\}$ can be evaluated to *true* or *false* based on the assumption that $x \bowtie y$. By an easy induction, we obtain:

▶ **Lemma 11.** *For all $\eta \in \{\varphi, \varphi_1, \ldots, \varphi_\ell\}$, $M \in \mathbb{MSC}(P, \Sigma')$, and events e of M:*

- $M, x \mapsto e \models \exists y.\eta$ *iff* $[\![\eta]\!]^{\bowtie}_{\lambda(e),(p',\sigma')}$ *is true for some* $\bowtie \in \Omega$ *and* $(p', \sigma') \in \mathsf{type}^{\bowtie}_M(e)$.
- $M, x \mapsto e \models \forall y.\eta$ *iff* $[\![\eta]\!]^{\bowtie}_{\lambda(e),(p',\sigma')}$ *is true for all* $\bowtie \in \Omega$ *and* $(p', \sigma') \in \mathsf{type}^{\bowtie}_M(e)$.

Therefore, we start by constructing a CFM $\mathcal{A}_{\mathsf{types}}$ that "labels" each event with its type. Formally, $\mathcal{A}_{\mathsf{types}}$ runs on *extended* MSCs, whose events have additional labels from the finite alphabet $\mathbb{T}_{P,\Sigma'}$. More generally, given a finite alphabet Γ, it will be convenient to consider a Γ-extended MSC from $\mathbb{MSC}(P, \Sigma' \times \Gamma)$, in the obvious way, as a pair (M, γ) where $M \in \mathbb{MSC}(P, \Sigma')$ and γ is a function from the set of events E to Γ.

▶ **Theorem 12.** *There is a CFM $\mathcal{A}_{\mathsf{types}}$ over P and $\Sigma' \times \mathbb{T}_{P,\Sigma'}$ with $2^{|\Sigma'| \cdot 2^{\mathcal{O}(|P| \log |P|)}}$ states such that $L(\mathcal{A}_{\mathsf{types}}) = \{(M, \mathsf{type}_M) \mid M \in \mathbb{MSC}(P, \Sigma')\}$.*

The proof of Theorem 12 is given in Section 4. Before this, we show that Theorem 12 (together with Lemmas 9 and 11) implies Theorem 7(2).

Proof of Theorem 7(2). We first convert the given sentence $\xi \in \mathrm{EMSO}^2[\rightarrow, \lhd, \leq]$ into the linear-size Scott normal form $\exists X_1 \ldots \exists X_m.\psi$ where $\psi = \forall x.\forall y.\varphi \wedge \bigwedge_{i=1}^{\ell} \forall x.\exists y.\varphi_i$ (Lemma 9). According to Lemma 11, the CFM for ψ is obtained from $\mathcal{A}_{\mathsf{types}}$ by restricting the transition relation: We keep a transition of process p with label $(\sigma, (\tau_{\bowtie})_{\bowtie \in \Omega}) \in \Sigma' \times \mathbb{T}_{P,\Sigma'}$ if $[\![\varphi]\!]^{\bowtie}_{(p,\sigma),(p',\sigma')}$ is *true* for all $\bowtie \in \Omega$ and $(p', \sigma') \in \tau_{\bowtie}$, and for all $1 \leq i \leq n$, $[\![\varphi_i]\!]^{\bowtie}_{(p,\sigma),(p',\sigma')}$ is *true* for some $\bowtie \in \Omega$ and $(p', \sigma') \in \tau_{\bowtie}$. Moreover, the new transition label will just be σ (the type is projected away). Finally, we project the extended alphabet $\Sigma' = \Sigma \times \{0,1\}^m$ to Σ. The resulting CFM $\mathcal{A}_\xi$ is equivalent to the given sentence ξ. ◀

4 CFM $\mathcal{A}_{\mathsf{types}}$ Checking the Type of Events

We obtain $\mathcal{A}_{\mathsf{types}}$ as the product of CFMs $\mathcal{A}^{\bowtie}$ over P and $\Sigma' \times 2^{P \times \Sigma'}$ such that $L(\mathcal{A}^{\bowtie}) = \{(M, \mathsf{type}^{\bowtie}_M) \mid M \in \mathbb{MSC}(P, \Sigma')\}$. Thus, it only remains to construct $\mathcal{A}^{\bowtie}$, for all $\bowtie \in \Omega$. The cases $\bowtie \in \{=, \rightarrow, \lhd, \rightarrow^{-1}, \lhd^{-1}\}$ are straightforward.

▶ **Lemma 13.** *There is a CFM $\mathcal{A}^{\ll^{-1}}$ over P and $\Sigma' \times 2^{P \times \Sigma'}$ with $2^{\mathcal{O}(|P \times \Sigma'|)}$ states such that $L(\mathcal{A}^{\ll^{-1}}) = \{(M, \mathsf{type}^{\ll^{-1}}_M) \mid M \in \mathbb{MSC}(P, \Sigma')\}$.*

Proof sketch. Consider Figure 1 and suppose $a, b, c \in \Sigma'$. At the time of reading event e, the CFM $\mathcal{A}^{\ll^{-1}}$ should deduce $\mathsf{type}^{\ll^{-1}}_M(e) = \{(p,a), (p,c), (r,a), (q,a), (q,b)\}$. To do so, it collects all labelings from $P \times \Sigma'$ that it has seen in the past, i.e., $\mathsf{type}^{\ll^{-1}}_M(e) \cup \mathsf{type}^{\rightarrow^{-1}}_M(e) \cup \mathsf{type}^{\lhd^{-1}}_M(e)$. Naively, one would then just remove the labels (p,a) and (q,b) of the predecessors f and g of e. However, this leads to the wrong result, since both (p,a) and (q,b) are contained in $\mathsf{type}^{\ll^{-1}}_M(e)$. In particular, there is another (q,b)-labeled event in $\mathcal{E}(e, \ll^{-1})$. The solution is to count the number of occurrences of each label up to 2. When reading e, the CFM will have seen (p,a) and (q,b) at least twice so that it can safely conclude that both are contained in $\mathsf{type}^{\ll^{-1}}_M(e)$. ◀

We then obtain $\mathcal{A}^{\ll}$ by symmetry. To complete the proof of Theorem 12, we construct below the CFM $\mathcal{A}^{\|}$ such that $L(\mathcal{A}^{\|}) = \{(M, \mathsf{type}^{\|}_M) \mid M \in \mathbb{MSC}(P, \Sigma')\}$.

For all $p, q \in P$ with $p \neq q$ and $a \in \Sigma'$, we define

$$L^0_{p,q,a} = \left\{(M,\gamma) \in \mathbb{MSC}(P, \Sigma' \times \{0,1\}) \mid \forall e \in E_p : \big(\gamma(e) = 0 \implies (q,a) \notin \mathsf{type}^{\|}_M(e)\big)\right\},$$
$$L^1_{p,q,a} = \left\{(M,\gamma) \in \mathbb{MSC}(P, \Sigma' \times \{0,1\}) \mid \forall e \in E_p : \big(\gamma(e) = 1 \implies (q,a) \in \mathsf{type}^{\|}_M(e)\big)\right\}.$$

▶ **Lemma 14.** *For all $p, q \in P$ with $p \neq q$ and $a \in \Sigma'$, there are CFMs $\mathcal{A}^0_{p,q,a}$ and $\mathcal{A}^1_{p,q,a}$ over P and $\Sigma' \times \{0,1\}$ with $2^{2^{\mathcal{O}(|P| \log |P|)}}$ states such that $L(\mathcal{A}^0_{p,q,a}) = L^0_{p,q,a}$ and $L(\mathcal{A}^1_{p,q,a}) = L^1_{p,q,a}$.*

Before proving Lemma 14, we explain how to derive the CFM $\mathcal{A}^{\|}$. Consider an extended MSC $(M, \tau) \in \mathbb{MSC}(P, \Sigma' \times \mathbb{T}_{P,\Sigma'})$. For all $p, q \in P$ with $p \neq q$ and $a \in \Sigma'$, define $\gamma^{M,\tau}_{p,q,a} \colon E \to \{0,1\}$ by $\gamma^{M,\tau}_{p,q,a}(e) = 1$ iff $e \in E_p$ and $(q,a) \in \tau_{\|}$ assuming $\tau(e) = (\tau_{\bowtie})_{\bowtie \in \Omega}$. From the CFMs $\mathcal{A}^0_{p,q,a}$ and $\mathcal{A}^1_{p,q,a}$, we easily derive a CFM $\mathcal{A}_{p,q,a}$ over P and $\Sigma' \times \mathbb{T}_{P,\Sigma'}$ such that $(M, \tau) \in L(\mathcal{A}_{p,q,a})$ iff $(M, \gamma^{M,\tau}_{p,q,a}) \in L^0_{p,q,a} \cap L^1_{p,q,a}$. We obtain $\mathcal{A}^{\|}$ as the intersection of the CFMs $\mathcal{A}_{p,q,a}$.

Construction of $\mathcal{A}^0_{p,q,a}$. Let $(M, \gamma) \in \mathbb{MSC}(P, \Sigma' \times \{0,1\})$ be an MSC. We can easily check that $(M, \gamma) \in L^0_{p,q,a}$ iff there is a path ν in M (i.e., a path in the directed graph $(E, \to \cup \lhd)$) such that all events e on process p with $\gamma(e) = 0$ and all events f such that $\lambda(f) = (q,a)$ are on ν. Therefore, the CFM $\mathcal{A}^0_{p,q,a}$ will try to guess such a path ν. This path is represented by a token moved along the MSC. Initially, exactly one process has the token. At each event, the automaton may choose to pass along the token to the next event of the current process, or (if the event is a write) to send the token to another process. Formally, (non)-possession of the token is represented by two states, s_{token} and $s_{\overline{\mathsf{token}}}$, and movements of the token from one process to another by messages. All global states are accepting. Notice that $\mathcal{A}^0_{p,q,a}$ has only two states per process. Process p may read an event labeled 0 only if it has the token, and process q may read a's only if it has the token. Clearly, $\mathcal{A}^0_{p,q,a}$ has an accepting run on (M, γ) iff $(M, \gamma) \in L^0_{p,q,a}$.

Construction of $\mathcal{A}^1_{p,q,a}$. Let $M = ((w_p)_{p \in P}, \lhd)$ be an MSC. For $e \in E$ and $F \subseteq E$, let $\|_p(e) = \{f \in E_p \mid f \parallel e\}$ and $\|_p(F) = \{e \in E_p \mid e \parallel f \text{ for some } f \in F\}$. Moreover, given $e \in E$, define $\downarrow_p(e) = \{f \in E_p \mid f < e\}$ and $\uparrow_p(e) = \{f \in E_p \mid e < f\}$. An *interval* in M is a (possibly empty) finite set of events $\{e_1, \ldots, e_k\}$ such that $e_1 \to \cdots \to e_k$. For all $e, f \in E_p$, we denote by $[e, f]$ the interval $\{g \in E_p \mid e \leq g \leq f\}$.

▶ Remark. For all $e \in E_q$ (recall that $p \neq q$), the sets $\downarrow_p(e)$, $\|_p(e)$, and $\uparrow_p(e)$ are intervals (possibly empty) of events on process p, such that $E_p = \downarrow_p(e) \uplus \|_p(e) \uplus \uparrow_p(e)$.

The idea is that $\mathcal{A}^1_{p,q,a}$ will guess a set of intervals covering all 1-labeled events on process p, and check that, for each interval I, there exists an event f such that $\lambda(f) = (q,a)$ and $I = \|_p(f)$. We first show that it will be sufficient for $\mathcal{A}^1_{p,q,a}$ to guess two sequences of disjoint intervals:

▶ **Lemma 15.** *Let $M = ((w_p)_{p \in P}, \lhd) \in \mathbb{MSC}(P, \Sigma')$ and $F = \{f \in E \mid \lambda(f) = (q,a)\}$. There exist subsets $F_1, F_2 \subseteq F$ such that the following hold:*

- $\|_p(F_1) \cup \|_p(F_2) = \|_p(F)$.
- *For $i \in \{1,2\}$, the intervals in $\|_p(F_i)$ are pairwise disjoint, and not adjacent: if $f, f' \in F_i$ and $f \neq f'$, then $\|_p(f) \cup \|_p(f')$ is not an interval.*

Proof. We first construct a set $F' \subseteq F$ by iteratively removing redundant events from F, i.e., until there remains no event f such that $\|_p(f) \subseteq \|_p(F' \setminus \{f\})$. Now, consider three events $f, f', f'' \in F'$ such that $f < f' < f''$. If $\|_p(f) \cup \|_p(f'')$ is an interval, we have $\|_p(f') \subseteq \|_p(f) \cup \|_p(f'')$, a contradiction with $f, f', f'' \in F'$.

Therefore, for each event $f \in F'$, there is at most one event $f' \in F'$ such that $f < f'$ and $\|_p(f) \cup \|_p(f')$ is an interval. We deduce that the set F' can be divided into two sets F_1 and F_2 satisfying the requirements of the lemma. ◀

So, $\mathcal{A}^1_{p,q,a}$ will proceed as follows. It will guess the sets F_1, F_2, $\|_p(F_1)$ and $\|_p(F_2)$, that is, label some events on process q with "F_1" or "F_2", and some events on process p with "F_1" and/or "F_2". This labeling must be such that on process q, only events initially labeled a may be labeled "F_1" or "F_2" (the sets guessed for F_1 and F_2 contain only events labeled a), and that on process p, all events initially labeled 1 must be labeled either "F_1", "F_2", or both (the sets guessed for $\|_p(F_1)$ and $\|_p(F_2)$ cover all events labeled 1 on process p). Then, for each $i \in \{1,2\}$, $\mathcal{A}^1_{p,q,a}$ will check condition C_i: for each non-empty maximal interval I of events marked F_i on process p, there exists an event f marked F_i on process q such that $I = \|_p(f)$. The CFM $\mathcal{A}^1_{p,q,a}$ will check C_1 and C_2 with two copies of a CFM $\mathcal{A}_{\mathsf{parallel}}$ defined below. But first, notice that if $\mathcal{A}^1_{p,q,a}$ has an accepting run on M then $M \in L^1_{p,q,a}$. Conversely, if $M \in L^1_{p,q,a}$, then if $\mathcal{A}^1_{p,q,a}$ guesses correctly the sets F_1, F_2, $\|_p(F_1)$ and $\|_p(F_2)$, it accepts.

The MSC language L_{parallel}. Define the language L_{parallel} of extended MSCs $(M,\zeta) \in \mathbb{MSC}(P, \Sigma' \times \{0,1\})$ with $\zeta\colon E \to \{0,1\}$ such that

- for each non-empty maximal interval I of 1-labeled events on process p, there exists exactly one 1-labeled event f on process q such that $\|_p(f) = I$,
- and conversely, for all 1-labeled events f on process q, there exists a non-empty maximal interval I of 1-labeled events on process p such that $\|_p(f) = I$.

We show below how to construct a CFM $\mathcal{A}_{\mathsf{parallel}}$ for the language L_{parallel}. Notice that $\mathcal{A}^1_{p,q,a}$ can check condition C_i ($i \in \{1,2\}$) by running $\mathcal{A}_{\mathsf{parallel}}$ on the MSC (M,ζ) where the events labeled 1 by ζ are those labeled F_i by $\mathcal{A}^1_{p,q,a}$.

We can decompose this problem one last time. Let Π (respectively, $\Pi_{p,q}$) be the set of process sequences $\pi = p_1 \dots p_n$ (respectively, with $p_1 = p$ and $p_n = q$) such that $n \geq 1$ and $p_i \neq p_j$ for $i \neq j$. For $\pi = p_1 \dots p_n \in \Pi$ and $e, f \in E$, we write $e \rightsquigarrow_\pi f$ if there exist events $e = e_1, f_1, e_2, f_2, \dots, e_n, f_n = f$ such that, for all i, we have $e_i, f_i \in E_{p_i}$, $e_i \to^* f_i$, and $f_i \lhd e_{i+1}$. For all events $e \in E$ such that $\{f \in E \mid f \rightsquigarrow_\pi e\}$ (respectively, $\{f \in E \mid e \rightsquigarrow_\pi f\}$) is non-empty, we let

$$\mathsf{pred}_\pi(e) = \max\{f \in E \mid f \rightsquigarrow_\pi e\} \quad \text{and} \quad \mathsf{succ}_\pi(e) = \min\{f \in E \mid e \rightsquigarrow_\pi f\}\,.$$

This is well-defined since all events in $\{f \in E \mid f \rightsquigarrow_\pi e\}$ (respectively, $\{f \in E \mid e \rightsquigarrow_\pi f\}$) are on the same process, hence are ordered. Note that, if $\pi = p$ consists of a single process, then, for all $e \in E_p$, we have $\mathsf{pred}_\pi(e) = e = \mathsf{succ}_\pi(e)$. Moreover, notice that $\leq = \bigcup_{\pi \in \Pi} \rightsquigarrow_\pi$.

- Let $L_{\mathsf{intervals}}$ be the set of MSCs (M,ζ) where the mapping $\zeta\colon E \to \{0,1\}$ defines (non-empty maximal) intervals $[e_1, e'_1], \dots, [e_k, e'_k]$ of 1-labeled events on process p and a sequence of 1-labeled events $f_1 < \dots < f_k$ on process q, such that, for all $1 \leq i \leq k$, we have $\downarrow_p(e_i) \subseteq \downarrow_p(f_i)$ and $\uparrow_p(e'_i) \subseteq \uparrow_p(f_i)$. This is illustrated in Figure 2.
- Let L_{left} be the set of all MSCs in $L_{\mathsf{intervals}}$ such that, for all $1 \leq i \leq k$ and $\pi \in \Pi_{p,q}$, if $\mathsf{pred}_\pi(f_i)$ is defined, then $\mathsf{pred}_\pi(f_i) < e_i$.
- Let L_{right} be the set of all MSCs in $L_{\mathsf{intervals}}$ such that, for all $1 \leq i \leq k$ and $\pi \in \Pi_{q,p}$, if $\mathsf{succ}_\pi(f_i)$ is defined, then $e'_i < \mathsf{succ}_\pi(f_i)$.

Note that $L_{\mathsf{parallel}} \subseteq L_{\mathsf{intervals}}$. The converse inclusion does not hold in general, since the intervals in MSCs from $L_{\mathsf{intervals}}$ may be too large. However, we have:

▶ **Lemma 16.** $L_{\mathsf{parallel}} = L_{\mathsf{left}} \cap L_{\mathsf{right}}$.

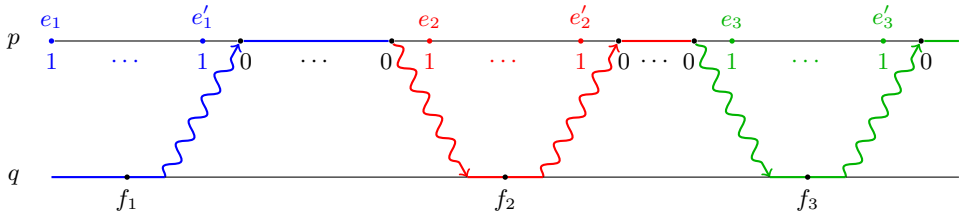


Figure 2 Constructions of $\mathcal{A}_{\mathsf{intervals}}$ and $\mathcal{A}_{\mathsf{parallel}}$.

Proof. Let $(M,\zeta) \in L_{\mathsf{parallel}}$ and $i \in \{1,\dots,k\}$. By definition, $[e_i, e'_i] = \|_p(f_i)$. Since $[e_i, e'_i]$ is non-empty, we have $\downarrow_p(e_i) = \downarrow_p(f_i)$ and $\uparrow_p(e'_i) = \uparrow_p(f_i)$. Hence, $(M,\zeta) \in L_{\mathsf{left}} \cap L_{\mathsf{right}}$.

Now, let $(M,\zeta) \in L_{\mathsf{left}} \cap L_{\mathsf{right}}$. Since $(M,\zeta) \in L_{\mathsf{intervals}}$, we have $\|_p(f_i) \subseteq [e_i, e'_i]$ for all i. Assume that there is $e \in [e_i, e'_i] \setminus \|_p(f_i)$. For instance $e \in \downarrow_p(f_i)$. Then, there exists $\pi \in \Pi_{p,q}$ such that $e \rightsquigarrow_\pi f_i$. Hence $e \le \mathsf{pred}_\pi(f_i)$, a contradiction with $(M,\zeta) \in L_{\mathsf{left}}$. ◀

A CFM for L_{parallel}. The last piece of the puzzle is a CFM $\mathcal{A}_{\mathsf{parallel}}$ such that $L(\mathcal{A}_{\mathsf{parallel}}) = L_{\mathsf{parallel}}$. It is built as the product (intersection) of CFMs $\mathcal{A}_{\mathsf{intervals}}$, $\mathcal{A}_{\mathsf{left}}$, and $\mathcal{A}_{\mathsf{right}}$.

▶ **Lemma 17.** *There is a CFM $\mathcal{A}_{\mathsf{intervals}}$ with a constant number of states such that we have $L(\mathcal{A}_{\mathsf{intervals}}) = L_{\mathsf{intervals}}$.*

Proof. Again, we implement a sort of token passing, which is illustrated in Figure 2. The token starts on process p iff the first p-event is labeled 0; otherwise, it must start on q. Similarly, the token ends on process p iff the last p-event is labeled 0; otherwise, it must end on q. Process p reads 0's when it holds the token, and 1's when it does not. Moreover, after sending the token, process p must read some 1-labeled events. When sent by p (respectively q), the token must reach q (respectively p) before returning to p (respectively q). Finally, process q reads only 0-labeled events when it does not hold the token. Moreover, process q checks that, within every maximal interval where it holds the token, there is exactly one 1-labeled event.

It is easy to check that $(M,\zeta) \in L_{\mathsf{intervals}}$ iff there exists a path along which the token is passed and satisfying the above conditions. ◀

We now show that there exists a CFM $\mathcal{A}_{\mathsf{left}}$ that accepts an MSC $(M,\zeta) \in L_{\mathsf{intervals}}$ iff $(M,\zeta) \in L_{\mathsf{left}}$. The idea is that $\mathcal{A}_{\mathsf{left}}$ guesses a coloring of the intervals of marked events such that checking $\mathsf{pred}_\pi(f_i) < e_i$ can be replaced with checking that $\mathsf{pred}_\pi(f_i)$ is not in an interval with the same color as $[e_i, e'_i]$. We need to prove that such a coloring exists, and that the colors associated with the $\mathsf{pred}_\pi(f_i)$ can be computed by the CFM.

▶ **Lemma 18.** *Let $(M,\zeta) \in L_{\mathsf{left}}$, let $[e_1, e'_1], \dots, [e_k, e'_k]$ be the sequence of maximal intervals of 1-labeled events on process p, and $f_1 < \cdots < f_k$ the corresponding events labeled 1 on process q. There exists a coloring $\chi\colon \{1,\dots,k\} \to \{1,\dots,|\Pi_{p,q}|+1\}$ such that, for all $i,j \in \{1,\dots,k\}$ and $\pi \in \Pi_{p,q}$, $\mathsf{pred}_\pi(f_j) \in [e_i, e'_i]$ implies $\chi(i) \neq \chi(j)$.*

Proof. We write $i \rightsquigarrow j$ when there exists $\pi \in \Pi_{p,q}$ such that $\mathsf{pred}_\pi(f_j) \in [e_i, e'_i]$. Notice that if $i \rightsquigarrow j$, then $i < j$ (otherwise, we would have $e_i \le \mathsf{pred}_\pi(f_j) < f_j < f_i$, hence, $e_i \le \mathsf{pred}_\pi(f_j) \le \mathsf{pred}_\pi(f_i)$, a contradiction with $(M,\zeta) \in L_{\mathsf{left}}$). So we can define χ by successively choosing colors for $1,\dots,k$: For all j, it suffices to choose a color $\chi(j) \in \{1,\dots,|\Pi_{p,q}|+1\}$ distinct from the at most $|\Pi_{p,q}|$ colors of indices $i < j$ such that $i \rightsquigarrow j$. ◀

▶ **Lemma 19.** *Let Θ be a finite set. There exists a (deterministic) CFM with $|\Theta|^{\mathcal{O}(|P|!)}$ states recognizing the set of doubly extended MSCs (M, θ, ξ) such that, for all events e, $\xi(e)$ is the partial function from Π to Θ such that $\xi(e)(\pi) = \theta(\mathsf{pred}_\pi(e))$.*

Proof. The CFM stores the label $\xi(e)$ of an event e in its state, and includes it in the message if e is a send event. At an event e on process u, the CFM checks that $\xi(e)(u) = \theta(e)$. Moreover, the CFM checks that:

- If e has no $\rightarrow$-predecessor and no $\lhd$-predecessor, then $\xi(e)(\pi)$ is undefined for all $\pi \neq u$.
- If e has one $\rightarrow$-predecessor f but no $\lhd$-predecessor, then $\xi(e)(\pi) = \xi(f)(\pi)$ for $\pi \neq u$.
- If e has one $\lhd$-predecessor g on process r, but no $\rightarrow$-predecessor, then $\xi(e)(\pi r u) = \xi(g)(\pi r)$, and $\xi(e)(\pi)$ is undefined if $\pi \neq u$ and π does not end with ru.
- If e has one $\rightarrow$-predecessor f and one $\lhd$-predecessor g on process r, then $\xi(e)(\pi r u) = \xi(g)(\pi r)$, and $\xi(e)(\pi) = \xi(f)(\pi)$ if $\pi \neq u$ and π does not end with ru. ◀

▶ **Lemma 20.** *There is a CFM $\mathcal{A}_{\mathsf{left}}$ with $2^{2^{\mathcal{O}(|P| \log |P|)}}$ states such that we have $L(\mathcal{A}_{\mathsf{left}}) \cap L_{\mathsf{intervals}} = L_{\mathsf{left}}$.*

Proof. Let $(M, \zeta) \in L_{\mathsf{intervals}}$ with $[e_1, e_1'], \ldots, [e_k, e_k']$ the non-empty maximal intervals of 1-labeled events on process p, and let $f_1 < \cdots < f_k$ be the corresponding 1-labeled events on process q. We can slightly modify $\mathcal{A}_{\mathsf{intervals}}$ so that on input (M, ζ), it guesses a coloring $\chi \colon \{1, \ldots, k\} \to \{1, \ldots, |\Pi_{p,q}| + 1\}$, and labels each event in $[e_i, e_i']$ with $\chi(i)$. The color of the upcoming interval $[e_i, e_i']$ is passed along with the token, so that at each f_i, the CFM has access to the color $\chi(i)$ (see Figure 2).

We can then compose that automaton with the CFM from Lemma 19, to compute, at each f_i and for all $\pi \in \Pi_{p,q}$ such that $\mathsf{pred}_\pi(f_i)$ is defined, the value $\zeta(\mathsf{pred}_\pi(f_i))$ and the color associated with $\mathsf{pred}_\pi(f_i)$. The CFM $\mathcal{A}_{\mathsf{left}}$ then checks that for all i and π, either $\mathsf{pred}_\pi(f_i)$ is undefined, or $\zeta(\mathsf{pred}_\pi(f_i)) = 0$, or the color associated with $\mathsf{pred}_\pi(f_i)$ is different from $\chi(i)$.

Suppose $(M, \zeta) \in L(\mathcal{A}_{\mathsf{left}}) \cap L_{\mathsf{intervals}}$. Then, for all i and π, $\mathsf{pred}_\pi(f_i)$ cannot be in an interval colored $\chi(i)$. In particular, this implies $\mathsf{pred}_\pi(f_i) \notin [e_i, e_i']$. Since $\uparrow_p(e_i') \subseteq \uparrow_p(f_i)$ and $\mathsf{pred}_\pi(f_i) \notin \uparrow_p(f_i)$, we deduce that $\mathsf{pred}_\pi(f_i) < e_i$ and $(M, \zeta) \in L_{\mathsf{left}}$. Conversely, suppose $(M, \zeta) \in L_{\mathsf{left}}$. Then, by Lemma 18, there exists a run in which the coloring guessed along the token passing is such that $\mathcal{A}_{\mathsf{left}}$ accepts. ◀

Finally, we obtain $\mathcal{A}_{\mathsf{parallel}}$ as the product (intersection) of $\mathcal{A}_{\mathsf{intervals}}$, $\mathcal{A}_{\mathsf{left}}$, and the "reversal" (or mirror) $\mathcal{A}_{\mathsf{right}}$ of $\mathcal{A}_{\mathsf{left}}$, which recognizes L_{right}. In fact, it is easy to see that CFMs are closed under reversal languages, in which both the process and the edge relations are inverted.

▶ **Lemma 21.** *There is a CFM $\mathcal{A}_{\mathsf{parallel}}$ with $2^{2^{\mathcal{O}(|P| \log |P|)}}$ states such that $L(\mathcal{A}_{\mathsf{parallel}}) = L_{\mathsf{parallel}}$.*

5 Conclusion

We showed that every EMSO^2 formula over MSCs can be effectively translated into an equivalent CFM of doubly exponential size, which is optimal. At the heart of our construction is a CFM $\mathcal{A}_{\mathsf{types}}$ of independent interest, which "outputs" the type of each event of an MSC. In particular, $\mathcal{A}_{\mathsf{types}}$ can be applied to other logics such as propositional dynamic logic (PDL), which combines modal operators and regular expressions [10]. It has been shown in [5] that every PDL formula can be translated into an equivalent CFM. We can extend this result by adding a modality $\langle \| \rangle$ to PDL, which "jumps" to some parallel event. For example, the formula $\neg \mathsf{E}(CS \wedge \langle \| \rangle CS)$ says that no two parallel events access a critical section. Note that

[5] considers infinite MSCs. However, it is easy to see that all our constructions can be extended to infinite MSCs.

A major open problem is whether every sentence from $\mathrm{FO}[\rightarrow, \lhd, \leq]$, with arbitrarily many variables, is equivalent to some CFM. To the best of our knowledge, the question is even open for the logic $\mathrm{FO}[\leq]$. Generally, it would be worthwhile to identify large classes of acyclic graphs of bounded degree such that all FO- or FO^2-definable languages (including the transitive closure of the edge relation) are "recognizable" (e.g., by a graph acceptor [25]).

References

1. R. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, 56(3):1–43, 2009.
2. N. Bedon. Logic and branching automata. *Logical Methods in Computer Science*, 11(4), 2015.
3. H. Björklund and T. Schwentick. On notions of regularity for data languages. *Theoretical Computer Science*, 411(4-5):702–715, 2010.
4. M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Transactions on Computational Logic*, 12(4):27, 2011.
5. B. Bollig, D. Kuske, and I. Meinecke. Propositional dynamic logic for message-passing systems. *Logical Methods in Computer Science*, 6(3:16), 2010.
6. B. Bollig and M. Leucker. Message-passing automata are expressively equivalent to EMSO logic. *Theoretical Computer Science*, 358(2-3):150–172, 2006.
7. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2), 1983.
8. J. Büchi. Weak second order logic and finite automata. *Z. Math. Logik, Grundlag. Math.*, 5:66–62, 1960.
9. C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98:21–52, 1961.
10. M. J. Fischer and R. E. Ladner. Propositional Dynamic Logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.
11. B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Information and Computation*, 204(6):920–956, 2006.
12. B. Genest, D. Kuske, and A. Muscholl. On communicating automata with bounded channels. *Fundamenta Informaticae*, 80(1-3):147–167, 2007.
13. E. Grädel and M. Otto. On logics with two variables. *Theor. Comput. Sci.*, 224(1-2):73–113, 1999.
14. M. Grohe and N. Schweikardt. Comparing the succinctness of monadic query languages over finite trees. In *Proceedings of CSL'03*, volume 2803 of *Lecture Notes in Computer Science*, pages 226–240. Springer, 2003.
15. W. Hanf. Model-theoretic methods in the study of elementary logic. In J. W. Addison, L. Henkin, and A. Tarski, editors, *The Theory of Models*. North-Holland, Amsterdam, 1965.
16. Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. *Distributed Computing*, 28(1):31–53, 2015. doi:10.1007/s00446-013-0202-3.
17. J. G. Henriksen, M. Mukund, K. Narayan Kumar, M. Sohoni, and P. S. Thiagarajan. A theory of regular MSC languages. *Information and Computation*, 202(1):1–38, 2005.
18. D. Kuske. Infinite series-parallel posets: Logic and languages. In *Proceedings of ICALP'00*, volume 1853 of *LNCS*, pages 648–662. Springer, 2000.

19 D. Kuske. Regular sets of infinite message sequence charts. *Information and Computation*, 187:80–109, 2003.

20 Antti Kuusisto. Modal logic and distributed message passing automata. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPIcs*, pages 452–468. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.CSL.2013.452.

21 Fabian Reiter. Distributed graph automata. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 192–201. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.27.

22 Fabian Reiter. Asynchronous distributed automata: A characterization of the modal mu-fragment. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 100:1–100:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.100.

23 J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

24 W. Thomas. On logical definability of trace languages. In *Proceedings of Algebraic and Syntactic Methods in Computer Science (ASMICS)*, Report TUM-I9002, Technical University of Munich, pages 172–182, 1990.

25 W. Thomas. Elements of an automata theory over partial orders. In *Proceedings of POMIV'96*, volume 29 of *DIMACS*. AMS, 1996.

26 B. A. Trakhtenbrot. Finite automata and monadic second order logic. *Siberian Math. J*, 3:103–131, 1962. In Russian; English translation in *Amer. Math. Soc. Transl.* 59, 1966, 23–55.

27 M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of LICS'86*, pages 332–344. IEEE Computer Society, 1986.

28 P. Weis. *Expressiveness and Succinctness of First-order Logic on Finite Words*. Ph.D. thesis, University of Massachusetts Amherst, 2011.

29 W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O. — Informatique Théorique et Applications*, 21:99–135, 1987.

On Approximating the Stationary Distribution of Time-reversible Markov Chains

Marco Bressan
Dipartimento di Informatica, Sapienza Università di Roma, Italy
bressan@di.uniroma1.it

Enoch Peserico
Dipartimento di Ingegneria dell'Informazione, Università di Padova, Italy
enoch@dei.unipd.it

Luca Pretto
Dipartimento di Ingegneria dell'Informazione, Università di Padova, Italy
pretto@dei.unipd.it

Abstract

Approximating the stationary probability of a state in a Markov chain through Markov chain Monte Carlo techniques is, in general, inefficient. Standard random walk approaches require $\tilde{O}(\tau/\pi(v))$ operations to approximate the probability $\pi(v)$ of a state v in a chain with mixing time τ, and even the best available techniques still have complexity $\tilde{O}(\tau^{1.5}/\pi(v)^{0.5})$; and since these complexities depend inversely on $\pi(v)$, they can grow beyond any bound in the size of the chain or in its mixing time. In this paper we show that, for time-reversible Markov chains, there exists a simple randomized approximation algorithm that breaks this "small-$\pi(v)$ barrier".

2012 ACM Subject Classification Mathematics of computing → Markov-chain Monte Carlo methods, Theory of computation → Sketching and sampling, Theory of computation → Random walks and Markov chains

Keywords and phrases Markov Chains, MCMC Sampling, Large Graph Algorithms, Randomized Algorithms, Sublinear Algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.18

Related Version For a full version of this paper see `https://arxiv.org/abs/1801.00196v1`, [7].

Funding Marco Bressan is supported in part by the SIR Grant RBSI14Q743. Enoch Peserico is supported in part by Univ. Padova under Project CAEPAE.

1 Introduction

We investigate the problem of approximating efficiently a *single* entry of the stationary distribution of an ergodic Markov chain. This problem has two main motivations. First, with the advent of massive-scale data, even complexities linear in the size of the input are often excessive [19]; therefore computing explicitly the entire stationary distribution, e.g. via the power method [10], can be simply infeasible. As an alternative one can then resort to approximating only individual entries of the vector, in exchange for a much lower computational complexity [13, 20]. In fact, if such a complexity is low enough one could efficiently "sketch" the whole vector by quickly getting a fair estimate of its entries. Second, in many practical cases one is really interested in just a few entries at a time. A classic example is that of network centralities, many of which are stationary distributions of an

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 18; pp. 18:1–18:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


ergodic Markov chain [4]. Indeed, the problem of approximating the Personalized PageRank score of a few nodes in a graph has been repeatedly addressed in the past [5, 6, 16, 15].

In this paper we seek for efficient algorithms for approximating the stationary probability $\pi(v)$ of some target state v in the state space of a discrete-time ergodic Markov chain. Besides the motivations above, the problem arises in estimating heat kernels and graph diffusions, testing the conductance of graphs and chains, developing local algorithms, and has applications in machine learning; see [3, 12] for a thorough discussion. We adopt a simple model where with a single operation one can either (i) simulate one step of the chain or (ii) retrieve the transition probability between a pair of states. Although recent research has provided encouraging results, existing algorithms suffer from a crucial bottleneck: to guarantee a small *relative* error in the approximation of $\pi(v)$, they incur a cost that grows with $1/\pi(v)$ itself (basically because estimating $\pi(v)$ via repeated sampling requires $1/\pi(v)$ samples). This is a crucial issue since in general there is no lower bound on $\pi(v)$; even worse, if the state space has n states, then most states have mass $\pi(v) = O(\frac{1}{n})$, and one can easily design chains where they have mass exponentially small in n. In general, then, the cost of existing algorithms can blow up far beyond $O(n)$ for almost all input states v. It is thus natural to ask if the dependence of the complexity on $\pi(v)$ is unavoidable. Unfortunately, one can easily show that $\Omega(\tau/\pi(v))$ operations can be necessary to estimate $\pi(v)$ within any constant multiplicative factor if one makes no assumption on the chain (see Appendix 5.2). To drop below this complexity barrier one must then necessarily look at special classes of Markov chains.

We present an algorithm that breaks this "small-$\pi(v)$ barrier" for *time-reversible* Markov chains. Time-reversible chains are a well-known subclass of Markov chains which lie at the heart of the celebrated Metropolis-Hastings algorithm [11] and are equivalent to random walks on weighted undirected graphs [14]. Formally, given any $\epsilon, \delta > 0$ and any state v in a time-reversible chain, our algorithm with probability $1 - \delta$ returns a multiplicative $(1 \pm \epsilon)$-approximation of $\pi(v)$ by using $\tilde{O}(\tau\|\pi\|^{-1})$ operations, where τ is the mixing time of the chain, $\|\cdot\|$ is the Euclidean norm and $\tilde{O}(\cdot)$ hides polynomials in ϵ^{-1}, $\ln(\delta^{-1})$, $\ln(\|\pi\|^{-1})$. The complexity is independent of $\pi(v)$, and for all but a vanishing fraction of states in the chain improves by factors at least $\sqrt{n}$ or $\sqrt{\tau}$ over previous algorithms. The heart of our algorithm is a randomized scheme for approximating the sum of a nonnegative vector by sampling its entries with probability proportional to their values. This scheme requires $\tilde{O}(\|\pi\|^{-1})$ samples if π is the distribution over the vector entries, which generalizes the $O(\sqrt{n})$ algorithm of [17] and is provably optimal. We prove that our algorithm for estimating $\pi(v)$ is essentially optimal as a function of τ, n and $\|\pi\|$; in fact one cannot do better even under a stronger computational model where all transition probabilities to/from all visited states are known. Finally, we show the number of distinct states visited by our algorithm may be further reduced, provided such a number satisfies some concentration hypotheses. This is useful if visiting a new state is expensive (e.g. if states are users in a social network). All our algorithms are simple to implement, require no tuning, and experimentally they appear faster than existing alternatives already for medium-sized chains.

The rest of the paper is organized as follows. Subsection 1.1 pins down definitions and notation; Subsection 1.2 formalizes the problem; Subsection 1.3 discusses related work; Subsection 1.4 summarizes our results. Section 2 presents our vector sum approximation algorithm. Section 3 presents our approximation algorithm for $\pi(v)$. The proofs of our lower bounds (Theorem 3 and Theorem 5), the pseudocode of our algorithms, and our experimental results can be found in the full version of this paper [7].

1.1 Preliminaries

A discrete-time, finite-state Markov chain is a sequence of random variables $X_0, X_1, \ldots$ taking value over a set of states $V = \{1, \ldots, n\}$, such that for all $i \geq 1$ and all $u_0, \ldots, u_i \in V$ with $\Pr(X_0 = u_0, \ldots, X_{i-1} = u_{i-1}) > 0$ we have $\Pr(X_i = u_i | X_0 = u_0, \ldots, X_{i-1} = u_{i-1}) = \Pr(X_i = u_i | X_{i-1} = u_{i-1})$. Denote by $\mathbf{P} = [p_{uu'}]$ the transition matrix of the chain, so that $p_{uu'} = \Pr(X_i = u' | X_{i-1} = u)$. We assume the chain is *ergodic*, and thus has a limit distribution that is independent from the distribution of X_0; the limit distribution then coincides with the *stationary distribution* π. Thus π is the unique distribution vector such that for any distribution vector π_0:

$$\pi = \pi\mathbf{P} = \lim_{t\to\infty} \pi_0\mathbf{P}^t \tag{1}$$

We denote by $\pi(u)$ the stationary probability, or *mass*, of u, and we always denote by v the target state whose mass is to be estimated. For any $V' \subseteq V$ we let $\pi(V')$ denote $\sum_{u\in V'} \pi(u)$. We also assume the chain is *time-reversible*, i.e. that for any pair of states u and u' we have:

$$\pi(u)p_{uu'} = \pi(u')p_{u'u} \tag{2}$$

We denote by τ the standard $\frac{1}{4}$-*mixing time* of the chain. In words, τ is the smallest integer such that after τ steps the total variation distance between π and the distribution of X_τ is bounded by $\frac{1}{4}$, irrespective of the initial distribution. Formally, $\tau := \min\{t : d(t) \leq \frac{1}{4}\}$, where

$$d(t) := \max_{\pi_0} \|\pi_0\mathbf{P}^t - \pi\|_{\mathrm{TV}} = \max_{\pi_0} \frac{1}{2}\|\pi_0\mathbf{P}^t - \pi\|_1 \tag{3}$$

After τ steps, the distribution of X_t converges to π exponentially fast; that is, if $t = \eta\tau$ with $\eta \geq 1$, then $\|\pi_0\mathbf{P}^t - \pi\|_{\mathrm{TV}} \leq 2^{-\eta}$. In the rest of the paper, $\|\cdot\|$ always denotes the ℓ^2 norm. One may refer to [14] for a detailed explanation of the notions recalled here.

Unless necessary, we drop multiplicative factors depending only on ϵ, δ (see below) from the asymptotic complexity notation. Furthermore, we use the tilde notation to hide polylogarithmic factors, i.e. we denote $O(f \cdot \mathrm{poly}(\log(f)))$ by $\tilde{O}(f)$.

1.2 Problem formulation

Consider now a discrete-time, finite-state, time-reversible, ergodic Markov chain on n states. The chain is initially unknown and can be accessed via two operations (also called queries):

step(): accepts in input a state u, and returns state u' with probability $p_{uu'}$

probe(): accepts in input a pair of states u, u', and returns $p_{uu'}$

These queries are the *de facto* model of previous work. *step*() is used in [5, 12, 6, 16, 15, 3] to simulate the walk, assuming each step costs $O(1)$. *probe*() is used in [16, 15, 3] to access the elements of the transition matrix, assuming again one access costs $O(1)$. Here, too, we assume *step*() and *probe*() as well as all standard operations (arithmetics, memory access, ...) cost $O(1)$. This includes set insertion and set membership testing; in case their complexity is $\omega(1)$, our bounds can be adapted correspondingly. The problem can now be formalized as follows. The algorithm is given in input a triple (v, ϵ, δ) where v is a state in the state space of the chain and ϵ, δ are two reals in $(0, 1)$. It must output a value $\hat{\pi}(v)$ such that, with probability $1 - \delta$, it holds $(1 - \epsilon)\pi(v) \leq \hat{\pi}(v) \leq (1 + \epsilon)\pi(v)$. The complexity of the algorithm is counted by the total number of operations it performs. Obviously we seek for an algorithm of minimal complexity.

A final remark. We say state u has been *visited* if $u = v$ or if u has been returned by a *step*() call. In line with previous work, we adopt the following "locality" constraint: the algorithm can invoke *probe*() and *step*() only on visited states.

1.3 Related work

Two recent works address precisely the problem of estimating $\pi(v)$ in Markov chains. The key differences with our paper are that they consider general (i.e. not necessarily time-reversible) chains, and that we aim at a small *relative* error for *any* $\pi(v)$ and not only for large $\pi(v)$.

- [12] gives a local approximation algorithm based on estimating return times via truncated random walks. Given any $\Delta > 0$, if $\pi(v) \geq \Delta$ the algorithm with probability $1-\delta$ outputs a multiplicative $\epsilon Z(v)$-approximation of $\pi(v)$, where $Z(v)$ is a "local mixing time" that depends on the structure of the chain. The cost is $\tilde{O}(\ln(1/\delta)/\epsilon^3\Delta)$ *step*() calls. If one wants a multiplicative $(1 \pm \epsilon)$-approximation of $\pi(v)$ for a generic v, the cost becomes $\tilde{O}(\tau/\pi(v))$ *step*() calls since one must wait for the walks to hit v after having mixed.
- [3] gives an algorithm to approximate ℓ-step transition probabilities based on coupling a local exploration of the transition matrix $\mathbf{P}$ with simulated random walks. Given any $\Delta > 0$, if the probability to be estimated is $\geq \Delta$ then with probability $1 - \delta$ the algorithm gives a multiplicative $(1 \pm \epsilon)$-approximation of it at an expected cost of $\tilde{O}(\ell^{1.5}\sqrt{d\,\ln(1/\delta)}\,/\,\epsilon\Delta^{0.5})$ calls to both *step*() and *probe*(), for a uniform random choice of v in the chain, where d is the density of $\mathbf{P}$. To estimate $\pi(v)$ for a generic v one must set $\ell = \tau$ and $\Delta = \pi(v)$, and since if the chain is irreducible then $d = \Omega(1)$, the bound stays at $\tilde{O}(\tau^{1.5}/\pi(v)^{0.5})$. This does not contradict our lower bound of Appendix 5.2, since their model allows for probing transition probabilities even between unvisited states.

Similar results are known for specific Markov chains, and in particular for PageRank (note that in PageRank $\tau = O(1)$). [5, 6] give an algorithm for approximating the PageRank $\pi(v)$ of the nodes v having $\pi(v) \geq \Delta$, at the cost of $\tilde{O}(1/\Delta)$ *step*() calls; again, if one desires a multiplicative $(1 \pm \epsilon)$-approximation of $\pi(v)$, the cost becomes $\tilde{O}(1/\pi(v))$. [16] gives an algorithm, with techniques similar to [3], for estimating the Personalized PageRank $\pi(v)$ of a node v; if one aims at a multiplicative $(1 \pm \epsilon)$-approximation of $\pi(v)$, the algorithm makes $\tilde{O}(d^{0.5}/\pi(v)^{0.5})$ *step*() and *probe*() calls where d is the average degree of the graph. Similar bounds can be found in [15] for Personalized PageRank on undirected graphs.

Summarizing, existing algorithms require either $\tilde{O}(\tau/\pi(v))$ or $\tilde{O}(\tau^{1.5}/\pi(v)^{0.5})$ *step*() and *probe*() calls to ensure a $(1 \pm \epsilon)$-approximation of $\pi(v)$ for a generic state v. Note that the complexity and approximation guarantees of these algorithms depend on knowledge of τ; our algorithms are no exception, and we prove our bounds as a function of τ.

Finally, for the problem of estimating the sum of a nonnegative n-entry vector $\mathbf{x}$ by sampling its entries x_i, with probability $\pi_i = x_i/\sum_i x_i$, the only algorithm existing to date is that of [17]. That algorithm takes $O(\sqrt{n})$ samples independently of π, while ours needs $O(\sqrt{n})$ samples only in the worst case, i.e. if π is (essentially) the uniform distribution.

1.4 Our results

Our first contribution is SumApprox, a randomized algorithm for estimating the sum γ of a nonnegative vector $\mathbf{x}$, assuming one can sample its entries according to the probability distribution $\pi = \mathbf{x}/\gamma$. Formally, we prove:

▶ **Theorem 1.** *Given any $\delta, \epsilon \in (0,1)$, SumApprox($\epsilon, \delta$) with probability at least $1-\delta$ returns a multiplicative $(1 \pm \epsilon)$-approximation of γ by taking $O\big(\|\pi\|^{-1}\epsilon^{-3}(\ln\frac{1}{\delta})^{3/2}\big)$ samples.*

SumApprox is extremely simple, yet it improves on the state-of-the-art $O(\sqrt{n})$ algorithm of [17]. We prove $\Omega\big(\|\pi\|^{-1}\big)$ samples are necessary, too, to get a fair estimate of γ.

We then employ SumApprox to build MassApprox, a randomized algorithm for approximating $\pi(v)$. Random-walk-based sampling and time reversibility are the ingredients that allow one to make the connection. We prove:

▶ **Theorem 2.** *Given any $\delta, \epsilon \in (0,1)$ and any state v in a time-reversible Markov chain, MassApprox(ϵ, δ, v) with probability $(1-\delta)$ returns a multiplicative $(1 \pm \epsilon)$ approximation of $\pi(v)$ using $\tilde{O}(\tau\|\pi\|^{-1}\epsilon^{-3}(\ln \frac{1}{\delta})^{3/2}) = \tilde{O}(\tau\|\pi\|^{-1})$ elementary operations and calls to step() and probe().*

Previous algorithms work also for general (i.e. non-reversible) chains; but on the $n-o(n)$ states with mass $\pi(v) = O(1/n)$, their complexity becomes at least $\tilde{O}(\tau n)$ [12] or $\tilde{O}(\tau^{1.5}\sqrt{n})$ [3]. In fact, $\pi(v)$ can be arbitrarily small (even exponentially small in n and τ) for almost all states in the chain, so for almost all states the complexity of previous algorithms blow up while that of MassApprox remains unchanged: since $\|\pi\|^{-1} \leq \sqrt{n}$ for any π, the complexity of MassApprox is at most $\tilde{O}(\tau\sqrt{n})$.

Next, we show that MassApprox is optimal as a function of τ, n and $\|\pi\|$, up to small factors. In fact, no algorithm can perform better even if equipped with an operation *neigh*(u) that returns all incoming and outgoing transition probabilities of u. Formally, we prove:

▶ **Theorem 3.** *For any function $\nu(n) \in \Omega(1/\sqrt{n}) \cap O(1)$ there is a family of time-reversible chains on n states where (a) $\|\pi\| = \Theta(\nu(n))$, and (b) there is a target state v such that, to estimate its mass $\pi(v)$ within any constant multiplicative factor with constant probability, any algorithm requires $\Omega(\tau\|\pi\|^{-1}/\ln n)$* neigh*() calls where τ is the mixing time of the chain.*

Although bounding time complexity is our primary goal, in some scenarios one wants to bound the *footprint*, i.e. the number of distinct states visited. Obviously, the footprint of MassApprox is bounded by its complexity (Theorem 2). We give a second algorithm, FullMassApprox, whose footprint can be smaller than that of MassApprox depending on τ, n, and $\|\pi\|$. More precisely, we prove a footprint bound that is conditional on the concentration of the footprint itself (see Subsection 3.1 for the intuition behind it).

▶ **Theorem 4.** *Let $N_{v,T}$ be the number of distinct states visited by a random walk of T steps starting from v. Assume for a function $\bar{\tau}$ of the chain we have $Pr[N_{v,T} \notin \Theta(\mathbb{E}[N_{v,T}])] = o\big(\frac{\bar{\tau}}{\mathbb{E}[N_{v,T}]}\big)$. Then, given any $\delta, \epsilon \in (0,1)$, with probability $(1-\delta)$ one can obtain a multiplicative $(1 \pm \epsilon)$-approximation of $\pi(v)$ by visiting $O(f(\epsilon,\delta)(\tau \ln n + \sqrt{\bar{\tau} n}))$ distinct states.*

If in Theorem 4 we have $\bar{\tau} = \tau$, then FullMassApprox is essentially optimal too. Formally:

▶ **Theorem 5.** *For any function $\tau(n) \in \Omega(\ln n) \cap O(n)$ there is a family of time-reversible chains on n states where (a) the mixing time is $\tau = \Theta(\tau(n))$, and (b) there is a target state v such that, to estimate its mass $\pi(v)$ within any constant multiplicative factor with constant probability, any algorithm requires $\Omega(\sqrt{\tau n/\ln n})$* neigh*() calls.*

2 Estimating sum by weighted sampling

In this section we analyse the following problem. We are given a vector of nonnegative reals γ_u indexed by the elements u of a set V. The vector is unknown, including its length, but we can draw samples from V according to the distribution π where u has probability $\gamma_u / \sum_{u \in V} \gamma_u$. The goal is to approximate the vector sum $\gamma = \sum_{u \in V} \gamma_u$. We describe a

simple randomized algorithm, SumApprox, which proceeds by repeatedly drawing samples and checking for repeats (i.e. a draw that yields an element already drawn before). The key intuition is the following: at any instant, if $S \subseteq V$ is the subset of elements drawn so far, then the next draw is a repeat with probability $\sum_{u \in S} \gamma_u / \gamma$. By drawing a sequence of samples we can thus flip a sequence of binary random variables, each one telling if a draw is a repeat, whose expectation is known save for the factor $1/\gamma$. If the sum of these random variables is sufficiently close to its expectation, one can then get a good approximation of γ by simply computing a ratio. The code of SumApprox is listed below.

Algorithm SumApprox(ϵ, δ).

1: $S \leftarrow \emptyset$ ▷ distinct elements drawn so far
2: $w_S \leftarrow 0$ ▷ $\sum_{u \in S} \gamma_u$ for the current S
3: $w \leftarrow 0$ ▷ cumulative sum of $\sum_{u \in S} \gamma_u$ so far
4: $r \leftarrow 0$ ▷ number of repeats so far
5: $k_{\epsilon,\delta} \leftarrow \lceil \frac{2+4.4\epsilon}{\epsilon^2} \ln \frac{3}{\delta} \rceil$ ▷ halting threshold on the number of repeats
6: **while** $r < k_{\epsilon,\delta}$ **do**
7: $\quad w \leftarrow w + w_S$
8: $\quad (u, \gamma_u) \leftarrow$ sample drawn from distribution π
9: $\quad$ **if** $u \in S$ **then** ▷ detect collision
10: $\quad\quad r \leftarrow r + 1$
11: $\quad$ **else**
12: $\quad\quad S \leftarrow S \cup \{u\}$
13: $\quad\quad w_S \leftarrow w_S + \gamma_u$
14: **return** w/r ▷ estimate of γ

We prove:

▶ **Theorem 6.** *SumApprox(ϵ, δ) with probability at least $1 - \frac{2\delta}{3}$ returns an estimate $\hat{\gamma}$ such that $|\hat{\gamma} - \gamma| < \epsilon\gamma$.*

Proof. We make use of a martingale tail inequality originally from [9] and stated (and proved) in the following form as Theorem 2.2 of [1], p. 1476:

▶ **Theorem 7** ([1], Theorem 2.2)**.** *Let $(Z_0, Z_1, \ldots)$ be a martingale with respect to the filter $(\mathcal{F}_i)$. Suppose that $Z_{i+1} - Z_i \leq M$ for all i, and write $V_t = \sum_{i=1}^{t} Var(Z_i|\mathcal{F}_{i-1})$. Then for any $z, v > 0$ we have*

$$Pr\big[Z_t \geq Z_0 + z, V_t \leq v \text{ for some } t\big] \leq \exp\Big[-\frac{z^2}{2(v + Mz)}\Big]$$

Let us plug into the formula of Theorem 7 the appropriate quantities from SumApprox:

- Let X_i be the $(i+1)^{th}$ sample (i.e. (X_i, γ_{X_i}) is the pair (u, γ_u) drawn at the $(i+1)^{th}$ invocation of line 8).
- Let $\mathcal{F}_i$ be the event space generated by $X_0, \ldots, X_i$, so that for any random variable Y, with $\mathbb{E}[Y|\mathcal{F}_i]$ we mean $\mathbb{E}[Y|X_0, \ldots, X_i]$ and with $Var[Y|\mathcal{F}_i]$ we mean $Var[Y|X_0, \ldots, X_i]$.
- Let $\chi_i = \mathbb{1}[X_i \in \bigcup_{j=0}^{i-1} \{X_j\}]$ be the indicator variable of a repeat on the $(i+1)^{th}$ sample.
- Let $P_i = \sum_{u \in \cup_{j=0}^{i-1} \{X_j\}} \frac{\gamma_u}{\gamma}$ be the probability of a repeat on the $(i+1)^{th}$ sample as a function of all the (distinct) samples up to the i^{th}, i.e. $P_i = \mathbb{E}[\chi_i|\mathcal{F}_{i-1}] \leq 1$.
- Let $Z_i = \sum_{j=0}^{i} (\chi_j - P_j)$; it is easy to see that $(Z_i)_{i \geq 0}$ is a martingale with respect to the filter $(\mathcal{F}_i)_{i \geq 0}$, since Z_i is obtained by adding to Z_{i-1} the indicator variable χ_i and subtracting P_i i.e. its expectation in $\mathcal{F}_{i-1}$. More formally, $\mathbb{E}[Z_i|\mathcal{F}_{i-1}] = \mathbb{E}[Z_{i-1} +$

$\chi_i - P_i|\mathcal{F}_{i-1}]$, and since Z_{i-1} and P_i are completely determined by $X_0, \ldots, X_{i-1}$, the right-hand term is simply $Z_{i-1} + (\mathbb{E}[\chi_i|\mathcal{F}_{i-1}] - P_i) = Z_{i-1}$. Note also that $Z_0 = 0$.

- Let $M = 1$, noting that $|Z_{i+1} - Z_i| = |\chi_{i+1} - P_{i+1}| \leq 1$ for all i.

Finally, note that $Var(Z_j|\mathcal{F}_{j-1}) = Var(\chi_j|\mathcal{F}_{j-1})$ (as $Z_j = Z_{j-1} + \chi_j - P_j$ and, again, Z_{j-1} and P_j are completely determined by $X_0, \ldots, X_{j-1}$). Since $Var(\chi_j|\mathcal{F}_{j-1}) = P_j(1-P_j) \leq P_j$, we have $V_i = \sum_{j=1}^{i} Var(Z_j|\mathcal{F}_{j-1}) \leq \sum_{j=1}^{i} P_j$. Theorem 7 then yields the following:

▶ **Lemma 8.** *For all $z, v > 0$ we have*

$$Pr\Big[Z_i \geq z, \sum_{j=1}^{i} P_j \leq v \text{ for some } i\Big] \leq \exp\Big[-\frac{z^2}{2(v+z)}\Big] \tag{4}$$

Recall now SumApprox. Note that $\sum_{j=1}^{i} P_j$ and Z_i are respectively the value of $\frac{w}{\gamma}$ and of $r - \frac{w}{\gamma}$ just after the *while* loop has been executed for the $(i+1)$-th time. Note also that, when SumApprox returns, $r = k_{\epsilon,\delta}$. Therefore the event that, when SumApprox returns, $\frac{w}{r} \leq \gamma(1-\epsilon)$ i.e. $\frac{w}{\gamma} \leq r(1-\epsilon) \leq (1-\epsilon)k_{\epsilon,\delta}$ corresponds to the event that $Z_i \geq \epsilon r = \epsilon k_{\epsilon,\delta}$ and $\sum_{j=1}^{i} P_j \leq (1-\epsilon)k_{\epsilon,\delta}$. Invoking Lemma 8 with $z = \epsilon k_{\epsilon,\delta}$ and $v = (1-\epsilon)k_{\epsilon,\delta}$:

$$\Pr\Big[\frac{w}{r} \leq \gamma(1-\epsilon)\Big] \leq \exp\Big[-\frac{\epsilon^2 k_{\epsilon,\delta}^2}{2(\epsilon k_{\epsilon,\delta} + (1-\epsilon)k_{\epsilon,\delta})}\Big] = \exp\Big[-\frac{\epsilon^2 k_{\epsilon,\delta}}{2}\Big] \tag{5}$$

which is smaller than $\delta/3$ since clearly $k_{\epsilon,\delta} > \frac{2}{\epsilon^2}\ln\frac{3}{\delta}$. Consider instead the event that, when SumApprox returns, $\frac{w}{r} \geq \gamma(1+\epsilon)$ i.e. $\frac{w}{\gamma} \geq r(1+\epsilon) = k_{\epsilon,\delta}(1+\epsilon)$. This is the event that $Z_i \leq -\epsilon k_{\epsilon,\delta}$, or equivalently $-Z_i \geq \epsilon k_{\epsilon,\delta}$. Note that Lemma 8 still holds if we replace Z_i with $-Z_i$, as $(-Z_i)_{i\geq 0}$ too is obviously a martingale with respect to the filter $(\mathcal{F}_i)_{i\geq 0}$, with $-Z_0 = 0$. Let then $i_0 \leq i$ be the smallest time such that $-Z_{i_0} \geq \epsilon k_{\epsilon,\delta}$. Since $|Z_j - Z_{j-1}| \leq 1$, it must be $-Z_{i_0} < \epsilon k_{\epsilon,\delta} + 1$. Also, since $\sum_{j=0}^{i} \chi_j$ is nondecreasing with i, then $\sum_{j=0}^{i_0} \chi_j \leq k_{\epsilon,\delta}$. It follows that $\sum_{j=1}^{i_0} P_j = -Z_{i_0} + \sum_{j=0}^{i_0} \chi_j \leq \epsilon k_{\epsilon,\delta} + 1 + k_{\epsilon,\delta} = (1+\epsilon)k_{\epsilon,\delta} + 1$. Invoking again Lemma 8 with $z = \epsilon k_{\epsilon,\delta}$ and $v = (1+\epsilon)k_{\epsilon,\delta} + 1$, we obtain:

$$\Pr\Big[\frac{w}{r} \geq \gamma(1+\epsilon)\Big] \leq \exp\Big[-\frac{\epsilon^2 k_{\epsilon,\delta}^2}{2((1+2\epsilon)k_{\epsilon,\delta} + 1)}\Big] \tag{6}$$

Note that $\frac{1}{k_{\epsilon,\delta}} < \frac{\epsilon^2}{2+4.4\epsilon} < 0.2\epsilon$ since $\epsilon \leq 1$; so $2((1+2\epsilon) + \frac{1}{k_{\epsilon,\delta}}) < 2 + 4.4\epsilon$, and since $k_{\epsilon,\delta} \geq \frac{2+4.4\epsilon}{\epsilon^2}\ln\frac{3}{\delta}$ the right-hand term is at most $\frac{\delta}{3}$. Finally, by a simple union bound the probability that $|\hat{\gamma} - \gamma| \geq \epsilon\gamma$ is at most $2\frac{\delta}{3}$, and the proof of Theorem 6 is complete. ◀

▶ **Theorem 9.** *SumApprox(ϵ, δ) draws at most $\lceil 45\|\pi\|^{-1}\epsilon^{-3}(\ln\frac{3}{\delta})^{3/2}\rceil$ samples with probability at least $1 - \frac{\delta}{3}$.*

Proof. We show that the probability that $s = \lceil 45\|\pi\|^{-1}\epsilon^{-3}(\ln\frac{3}{\delta})^{3/2}\rceil$ draws yield less than $k_{\epsilon,\delta}$ repeats is less than $\frac{\delta}{3}$. Let $\bar{p} = \frac{5}{18}\|\pi\|\epsilon(\ln\frac{3}{\delta})^{-1/2}$. We consider two cases.

Case 1: $\exists u \in V$ with $\pi(u) > \bar{p}$. Let then C_u^s be the random variable counting the number of times u appears in s draws. Since if $C_u^s > k_{\epsilon,\delta}$ then u causes at least $k_{\epsilon,\delta}$ repeats, the probability that SumApprox needs more than s draws is upper bounded by $\Pr[C_u^s \leq k_{\epsilon,\delta}]$. Now $\mathbb{E}[C_u^s] = s\pi(u) > s\bar{p} > 45\frac{5}{18}\frac{1}{\epsilon^2}\ln\frac{3}{\delta} = \frac{12.5}{\epsilon^2}\ln\frac{3}{\delta} \geq 1.7(\frac{6.4}{\epsilon^2}\ln\frac{3}{\delta} + 1) \geq 1.7\lceil\frac{2+4.4\epsilon}{\epsilon^2}\ln\frac{3}{\delta}\rceil = 1.7k_{\epsilon,\delta}$, therefore $C_u^s \leq k_{\epsilon,\delta}$ implies $C_u^s < \frac{1}{1.7}\mathbb{E}[C_u^s] < (1-0.41)\mathbb{E}[C_u^s]$. Since C_u^s is a sum of independent binary random variables, the bounds of Appendix 5.1 give $\Pr[C_u^s \leq k_{\epsilon,\delta}] < \exp\big(-\frac{1}{2}0.41^2\mathbb{E}[C_u^s]\big) < \exp\big(-0.5\cdot 0.41^2\cdot\frac{12.5}{\epsilon^2}\ln\frac{3}{\delta}\big) < \exp\big(-1.05\ln\frac{3}{\delta}\big) < \frac{\delta}{3}$.

Case 2: $\pi(u) \le \bar{p}$ for all $u \in V$. Let then $\bar{s} = \lceil \bar{p}^{-1} \rceil$, let $\bar{S}$ be the set of distinct elements in the first $\bar{s}$ draws, and let $w(\bar{s}) = \sum_{u \in \bar{S}} \pi(u)$. First we show that $\mathbb{E}[w(\bar{s})] \ge \frac{4}{9}\bar{s}\|\pi\|^2$. Write $\mathbb{E}[w(\bar{s})] = \sum_{u \in V} \pi(u)(1 - (1-\pi(u))^{\bar{s}})$. Since for all $x \in [0,1]$ and $k \ge 1$ it holds $(1-x)^k \le (1+kx)^{-1}$, by setting $x = \pi(u)$ and $k = \bar{s}$ we obtain $1 - (1-\pi(u))^{\bar{s}} \ge 1 - (1+\bar{s}\pi(u))^{-1} = \bar{s}\pi(u)(1+\bar{s}\pi(u))^{-1}$. Moreover note that $\bar{p}^{-1} \ge \frac{18}{5} = 3.6$ and thus $\lceil \bar{p}^{-1} \rceil \le \frac{5}{4}\bar{p}^{-1}$. Therefore $\bar{s}\pi(u) \le \lceil \bar{p}^{-1} \rceil \bar{p} \le \frac{5}{4}$ for all u, and thus $\bar{s}\pi(u)(1+\bar{s}\pi(u))^{-1} \ge \bar{s}\pi(u)\frac{1}{1+\frac{5}{4}} = \frac{4}{9}\bar{s}\pi(u)$. Therefore $\mathbb{E}[w(\bar{s})] \ge \frac{4}{9}\bar{s}\sum_{u \in V}\pi(u)^2 = \frac{4}{9}\bar{s}\|\pi\|^2$. Now we consider two cases. First, suppose the event $w(\bar{s}) \ge 0.4\,\mathbb{E}[w(\bar{s})]$ takes place. For $i = \bar{s}+1, \dots, s$ let χ_i be the indicator random variable of the event that the i-th draw is an element of $\bar{S}$, and let $C_s = \sum_{i=\bar{s}+1}^{s} \chi_i$. Clearly SumApprox witnesses at least C_s repeats in the last $s - \bar{s}$ draws, and thus overall. We shall then bound $\Pr[C_s < k_{\epsilon,\delta}]$. First, since by hypothesis the total mass of $\bar{S}$ is $w(\bar{s}) \ge 0.4\,\mathbb{E}[w(\bar{s})]$, we also have $\mathbb{E}[\chi_i] \ge 0.4\,\mathbb{E}[w(\bar{s})] \ge \frac{1.6}{9}\bar{s}\|\pi\|^2$. Therefore $\mathbb{E}[C_s] = \sum_{i=\bar{s}+1}^{s}\mathbb{E}[\chi_i] \ge \frac{1.6}{9}(s-\bar{s})\bar{s}\|\pi\|^2$. Now note that $s - \bar{s} > 10\bar{s}$, therefore $\mathbb{E}[C_s] \ge \frac{16}{9}\bar{s}^2\|\pi\|^2$. Finally, since $\bar{s} = \lceil \bar{p}^{-1} \rceil \ge \frac{18}{5}\|\pi\|^{-1}\epsilon^{-1}(\ln\frac{3}{\delta})^{1/2}$, it holds $\mathbb{E}[C_s] \ge (\frac{18}{5})^2\frac{16}{9}\frac{1}{\epsilon^2}\ln\frac{3}{\delta} > 23\frac{1}{\epsilon^2}\ln\frac{3}{\delta} > 3.14k_{\epsilon,\delta}$. It follows that the event $C_s < k_{\epsilon,\delta}$ implies $C_s < \frac{1}{3.14}\mathbb{E}[C_s] < (1-0.68)\mathbb{E}[C_s]$. By the concentration bounds of Appendix 5.1, the probability of the latter is $\Pr[C_s < k_{\epsilon,\delta}] \le \exp\big(-\frac{1}{2}0.68^2\,\mathbb{E}[C_s]\big) < \exp\big(-\frac{1}{2}0.68^2\,23\frac{1}{\epsilon^2}\ln\frac{3}{\delta}\big) < \exp\big(-5\ln\frac{3}{\delta}\big) < \frac{\delta}{243}$. The second case corresponds to the event $w(\bar{s}) < 0.4\,\mathbb{E}[w(\bar{s})] = (1-0.6)\mathbb{E}[w(\bar{s})]$, of which we shall bound the probability. Let $\chi_u^{\bar{s}}$ be the indicator variable of the event $u \in \bar{S}$, so $w(\bar{s}) = \sum_{u \in V}\chi_u^{\bar{s}}\,\pi(u)$. Since $\pi(u) \le \bar{p}$ for all u, we can write $w(\bar{s}) = \bar{p}\sum_{u \in V}\chi_u^{\bar{s}}\,\bar{p}^{-1}\pi(u)$ so that the coefficients $\bar{p}^{-1}\pi(u)$ are in $[0,1]$. Clearly, the $\chi_u^{\bar{s}}$ are non-positively correlated. We can thus apply the bounds of Appendix 5.1 and get $\Pr[w(\bar{s}) < 0.4\,\mathbb{E}[w(\bar{s})]] \le \exp\big(-0.5\cdot 0.6^2\,\bar{p}^{-1}\mathbb{E}[w(\bar{s})]\big)$. By replacing the definitions and bounds for $E[w(\bar{s})]$, $\bar{s}$ and $\bar{p}^{-1}$ from above, we get $\Pr[w(\bar{s}) < 0.4\,\mathbb{E}[w(\bar{s})]] < \exp\big(-2.88\ln(\frac{3}{\delta})\big) < \frac{\delta}{23}$. Again by a union bound, the probability that SumApprox draws more than s samples is less than $\frac{\delta}{243} + \frac{\delta}{23} < \frac{\delta}{3}$. ◀

We remark that the previous existing algorithm for the sum estimation problem [17] needs knowledge of $n = |V|$ and uses $O(\sqrt{n}\epsilon^{-7/2}\log(n)(\log\frac{1}{\delta} + \log\frac{1}{\epsilon} + \log\log n))$ samples. SumApprox is simpler, oblivious to n, and gives more general bounds. It is also asymptotically faster unless π is (essentially) the uniform distribution.

Finally, we show that SumApprox is essentially optimal, by proving $\Omega(\|\pi\|^{-1})$ samples are in general necessary to estimate γ even if n is known in advance. This extends to arbitrary distributions the $\Omega(\sqrt{n})$ lower bound given by [17] for the uniform distribution.

▶ **Theorem 10.** *For any function $\nu(n) \in \Omega(n^{-\frac{1}{2}}) \cap O(1)$ there exist vectors $\mathbf{x} = \gamma\pi = (\gamma_1, \dots, \gamma_n)$ with $\|\pi\| = \Theta(\nu(n))$ such that $\Omega(\|\pi\|^{-1})$ samples are necessary to estimate γ within constant multiplicative factors with constant probability, even if n is known.*

Proof. Let $k \in \Theta(\nu(n)^{-2})$ with $1 \le k \le \frac{n}{2}$. Consider the two vectors $\mathbf{x} = (\gamma_1, \dots, \gamma_n)$ and $\mathbf{x}' = (\gamma'_1, \dots, \gamma'_n)$ defined as follows:

$$\gamma_j = 1 : j \le k, \qquad \gamma_j = \sqrt{k}/n : j > k$$
$$\gamma'_j = 1 : j \le 2k \qquad \gamma'_j = \sqrt{k}/n : j > 2k$$

Now let $\gamma = \sum_{i=1}^{n}\gamma_i$ and $\gamma' = \sum_{i=1}^{n}\gamma'_i$. One can check that $\gamma \le 2k$ and $|\gamma - \gamma'| \ge \frac{k}{2}$. Hence, to obtain an estimate $\hat{\gamma}$ of γ with $\hat{\gamma} \le \frac{5}{4}\gamma$, one must distinguish $\mathbf{x}$ from $\mathbf{x}'$. Note that the norm of $\pi = \mathbf{x}/\gamma$ is in $\Theta(1/\sqrt{k}) = \Theta(\nu(n))$, as requested. Now, for each one of $\mathbf{x}$ and $\mathbf{x}'$ in turn, pick a permutation of $\{1, \dots, n\}$ uniformly at random and apply it to the entries of the vector. Suppose then we sample $o(\|\pi\|^{-1}) = o(\sqrt{k})$ entries from $\mathbf{x}$. We shall see that, with probability $1 - o(1)$, we cannot distinguish $\mathbf{x}$ from $\mathbf{x}'$. First, note that the total mass

of the entries with value $\sqrt{k}/n$ is at most $1/\sqrt{k}$. Hence the probability of drawing *any* of those entries with $o(\sqrt{k})$ samples is $o(1)$, and we can assume all draws yield entries having value 1. Since there are $O(k)$ such entries in total, the probability of witnessing any repeat is also $o(1)$, and we can assume no repeat is witnessed. Furthermore, because of the random permutation, the indices of samples are distributed uniformly over $\{1,\dots,n\}$ (recall that we actually sample from the index set $\{1,\dots,n\}$, so we could use the distribution of the indices to distinguish $\mathbf{x}$ from $\mathbf{x}'$). The same argument applies to $\mathbf{x}'$, so drawing $o(\sqrt{k})$ samples from $\mathbf{x}'$ yields exactly the same distribution and the two vectors are indistinguishable. To adapt the construction to larger approximation factors, set $\gamma'_j = 1 : j \leq \eta k$ for η large enough. ◀

3 Approximating the stationary distribution

In this section we address the problem of approximating $\pi(v)$. Such a problem can in fact be reduced to the sum estimation problem of Section 2 by drawing states via random walks. The crux is determining how long the walks must be in order for the samples to come from a distribution close enough to π, so that the approximation guarantees of SumApprox transfer directly to our estimate of $\pi(v)$.

Consider a random walk of length $t+1$ that starts at v. Obviously we can simulate such a walk by setting $u_0 = v$ and then invoking $step(u_i)$ to obtain the state u_{i+1}, for $i = 0,\dots,t-1$. Crucially, using the time-reversibility of the chain, for any visited state u we can obtain the ratio γ_u between $\pi(u)$ and $\pi(v)$ using $O(1)$ operations. Formally, let $\gamma_v = \pi(v)/\pi(v) = 1$, and in general let $\gamma_u = \pi(u)/\pi(v)$. Note that:

$$\gamma_{u_{i+1}} = \frac{\pi(u_{i+1})}{\pi(v)} = \frac{\pi(u_{i+1})}{\pi(u_i)} \cdot \frac{\pi(u_i)}{\pi(v)} = \frac{\pi(u_{i+1})}{\pi(u_i)} \cdot \gamma_{u_i} \tag{7}$$

The time-reversibility of the chain (see Equation 2) implies $\frac{\pi(u_{i+1})}{\pi(u_i)} = \frac{p_{u_i,u_{i+1}}}{p_{u_{i+1},u_i}} = \frac{probe(u_i,u_{i+1})}{probe(u_{i+1},u_i)}$, hence we can compute $\gamma_{u_{i+1}}$ with $O(1)$ operations if we know γ_{u_i}. But then we can keep track of γ_u for any u visited so far, starting with $\gamma_{u_0} = 1$ and computing $\gamma_{u_{i+1}}$ by Equation 7 the first time u_{i+1} is visited.

Suppose now to pick t large enough so that the chain reaches its stationary distribution, i.e. $u_t \sim \pi$ irrespective of v. One is then drawing state u, as well as its associate weight γ_u, with probability $\pi(u)$. Now if we let $\gamma = \sum_{u\in V} \gamma_u$, then $\pi(u) = \gamma_u \gamma^{-1}$ and in particular $\pi(v) = \gamma^{-1}$. Therefore approximating $\pi(v)$ amounts to approximating γ; more formally, for any $\epsilon \in (0,1)$, if $\hat{\gamma}$ is a $(1 \pm \frac{\epsilon}{2})$-approximation of γ then $\hat{\gamma}^{-1}$ is a $(1 \pm \epsilon)$-approximation of $\pi(v)$. We can therefore reduce to the sum approximation problem of Section 3: compute with probability $(1-\delta)$ a $(1 \pm \epsilon)$-approximation of γ, assuming we can draw pairs (u, γ_u) according to π. The only problem is that by simulating the chain we can only come close to (but not exactly on) the stationary distribution π. We must then tie the approximation guarantees of SumApprox to the length t of the random walks, or better to the distance $\|\pi' - \pi\|_{\mathrm{TV}}$ between π and the distribution π' from which u_t is drawn. Formally, we show:

▶ **Lemma 11.** *There exists some constant $c > 0$ such that the following holds. Choose any $\delta, \epsilon \in (0,1)$, and suppose we draw the pairs (u, γ_u) from a distribution π' such that $\|\pi - \pi'\|_{\mathrm{TV}} \leq \left(\frac{\epsilon\|\pi\|}{\ln(3/\delta)}\right)^c$. Then SumApprox$(\frac{\epsilon}{2}, \delta)$ with probability at least $1-\delta$ returns a multiplicative $(1 \pm \epsilon)$-approximation of γ by taking at most $\lceil 720\|\pi\|^{-1}\epsilon^{-3}(\ln\frac{3}{\delta})^{3/2}\rceil$ samples.*

Proof. Let us start with the bound on the number of samples. Recall the proof of Theorem 9, and note that the whole argument depends on π but not on the values γ_u. Indeed, π alone determines the probability of repeats and thus controls the distribution of the number of

samples drawn by SumApprox. Hence, by Theorem 9 SumApprox$(\frac{\epsilon}{2},\delta)$ takes more than $\lceil 45\|\pi'\|^{-1}8\epsilon^{-3}(\ln\frac{3}{\delta})^{3/2}\rceil = \lceil 360\|\pi'\|^{-1}\epsilon^{-3}(\ln\frac{3}{\delta})^{3/2}\rceil$ samples with probability less than $\frac{\delta}{3}$. Now $\|\pi-\pi'\| \le 2\|\pi-\pi'\|_{\mathrm{TV}} \le 2\big(\frac{\epsilon\|\pi\|}{\ln(3/\delta)}\big)^c \le \|\pi\|2(\ln 3)^{-c}$, which for $c \ge 15$ is bounded by $\frac{1}{2}\|\pi\|$. Then, since $\|\pi'\| \ge \|\pi\| - \|\pi-\pi'\|$, we have $\|\pi'\|^{-1} \le 2\|\pi\|^{-1}$ and the bound above is in turn bounded by $\lceil 720\|\pi\|^{-1}\epsilon^{-3}(\ln\frac{3}{\delta})^{3/2}\rceil$.

Let us now see the approximation guarantees. Recall the proof of Theorem 6. We want to show again that $\Pr[|\frac{w(s)}{r}-\gamma| \ge \frac{\epsilon}{2}\gamma] \le \frac{2\delta}{3}$. However, now the samples are drawn according to π' instead of π. Let then $P'_j = \sum_{u\in\cup_{h=0}^{j-1}\{X_h\}}\pi'(u)$ and $Z'_i = \sum_{j=0}^{i}(\chi_j - P'_j)$; in a nutshell, P'_j and Z'_i are the analogous of P_j and Z_i under π'. It is immediate to check that Lemma 8 holds with Z'_i and P'_j in place of Z_i and P_j. Let now $w'(i) = \gamma\sum_{j=1}^{i}P'_j$. Note that $\sum_{j=1}^{i}P'_j$ and Z'_i are respectively the value of $\frac{w'(i)}{\gamma}$ and of $r - \frac{w'(i)}{\gamma}$ just after line 9 has been executed for the $(i+1)$-th time. Therefore, the argument following Lemma 8 holds if we put $w'(i)$ in place of the value taken by w after the $(i+1)$-th execution of line 9. Hence the same bounds hold, and SumApprox$(\frac{\epsilon}{2},\delta)$ ensures $\Pr[|\frac{w'(s)}{r}-\gamma| \ge \frac{\epsilon}{2}\gamma] \le \frac{\delta}{3}$ where s is the total number of draws. Now note that SumApprox does not return $\frac{w'(s)}{r}$, but $\frac{w(s)}{r}$ where $w(i) = \gamma\sum_{j=1}^{i}P_j$ is the value of w in SumApprox after line 9 has been executed for the $(i+1)$-th time. We shall now make $|\frac{w(s)}{r} - \frac{w'(s)}{r}| \le \frac{\epsilon}{2}\gamma$; by the triangle inequality we will then be done. First of all, by the definition of $w(s)$ and $w'(s)$ we have

$$\Big|\frac{w(s)}{r} - \frac{w'(s)}{r}\Big| = \gamma r^{-1}\Big|\sum_{j=1}^{s}P_j - \sum_{j=1}^{s}P'_j\Big| \le \gamma r^{-1}\sum_{j=1}^{s}\big|P_j - P'_j\big| \tag{8}$$

Now note that $|P_j - P'_j| \le \|\pi-\pi'\|_{\mathrm{TV}}$, since P_j and P'_j are the probability of the same event under respectively π and π'. Therefore the right-hand side of Equation 8 is bounded by $\gamma r^{-1}s\,\|\pi-\pi'\|_{\mathrm{TV}}$. Now, when SumApprox$(\frac{\epsilon}{2},\delta)$ terminates $r = k_{\frac{\epsilon}{2},\delta} \ge 4\frac{2+2.2\epsilon}{\epsilon^2}\ln\frac{3}{\delta}$, and by hypothesis $\|\pi-\pi'\|_{\mathrm{TV}} \le \big(\frac{\epsilon\|\pi\|}{\ln(3/\delta)}\big)^c$. Therefore:

$$\Big|\frac{w(s)}{r} - \frac{w'(s)}{r}\Big| \le \gamma s\,\frac{\epsilon^2}{4(2+2.2\epsilon)\ln(\frac{3}{\delta})}\Big(\frac{\epsilon\|\pi\|}{\ln(\frac{3}{\delta})}\Big)^c \le \gamma\, s\,\|\pi\|\,\frac{\epsilon^{3+c}}{8\ln(\frac{3}{\delta})^{1+c}} \tag{9}$$

Finally, recall from above that with probability $1-\frac{3}{\delta}$ we have $s \le \lceil 720\|\pi\|^{-1}\epsilon^{-3}(\ln\frac{3}{\delta})^{3/2}\rceil$. In this case the equation above yields $|\frac{w(s)}{r} - \frac{w'(s)}{r}| \le \gamma\cdot 721\epsilon^c\ln(\frac{3}{\delta})^{0.5-c}$, which is smaller than $\frac{\epsilon}{2}\gamma$ for $c \ge \frac{1}{2} + \frac{\ln 1442}{\ln\ln 3} \approx 78$. A simple union bound completes the proof. ◀

We are now ready to prove Theorem 2. Pick $t = \tau\, c\ln(\|\pi\|^{-1}\epsilon^{-1}\ln\frac{3}{\delta})/\ln 2$, where c is the constant of Lemma 11 and τ is the mixing time of the chain. Simulate the walk for t steps starting from v, and let π' be the distribution of the final state. By the properties of the mixing time (see Section 1.1) it holds $\|\pi-\pi'\|_{\mathrm{TV}} \le 2^{-c\ln(\|\pi\|^{-1}\epsilon^{-1}\ln\frac{3}{\delta})/\ln 2} \le \big(\frac{\epsilon\|\pi\|}{\ln(3/\delta)}\big)^c$ and therefore by Lemma 11 we obtain a $(1\pm\epsilon)$ approximation of γ. By choosing ϵ small enough we can obtain a $(1\pm\epsilon')$ approximation of $\pi(v)$ for any desired ϵ'. The total number of operations performed is clearly bounded by $t = \tau\, c\ln(\|\pi\|^{-1}\epsilon^{-1}\ln\frac{3}{\delta})/\ln 2$ times the number of samples taken by SumApprox, and by substituting this value in the bound of Theorem 1 we obtain Theorem 2.

3.1 Reducing the footprint

In this section we describe FullMassApprox, the algorithm behind the bounds of Theorem 5. FullMassApprox is derived from MassApprox as follows. First, instead of performing a new walk of length t from v for each sample, the algorithm performs one long random walk of

length T and takes one sample every t steps. The correctness guarantees do not change, since although the samples do not come all from the same distribution, they are still drawn from a distribution sufficiently close to π. Second, after checking if the current draw yields a repeat, the algorithm includes in the set S not only the draw but also all other states visited so far. Again, this does not affect the guarantees, since we do not need the set S to be built on independent samples. However, this makes the mass of S grow potentially faster, so we can hope to get more repeats and decrease the total number of samples.

The concentration hypothesis. Before continuing to the proof of Theorem 5, let us provide some intuition behind the concentration hypothesis. Suppose the walk runs for $T = k\bar{\tau}$ steps for some $\bar{\tau} = \tau \operatorname{poly}(\log(\|\pi\|^{-1}))$. Such a process can be seen as a coupon collector over k rounds, where a subset of at most $\bar{\tau}$ states is collected (i.e. visited) at each round. Now, if we pick $\bar{\tau}' \leq \bar{\tau}$ with $\bar{\tau}' = \tau \operatorname{poly}(\log(\|\pi\|^{-1}))$, then in each round the $\bar{\tau} - \bar{\tau}'$ central steps are essentially independent of other rounds (more formally, the correlation is $O(\operatorname{poly}(n)^{-1})$). Each round is then in large part independent of the others; the issue is that the states visited *within* a single round are correlated. Such a correlation is responsible for the factor $\bar{\tau}$ in the concentration hypothesis and amounts for the (intuitive) fact that conditioning on the outcome of one step of the walk does not affect the distribution of those steps that are more than $\bar{\tau}$ steps away. We note that the concentration bounds of [8] give $\Pr[\sum_{i=1}^{T} f_i \notin (1 \pm \bar{\epsilon})\mathbb{E}[\sum_{i=1}^{T} f_i]] < 2\exp -\Omega\big(\bar{\epsilon}^2 \mathbb{E}[\sum_{i=1}^{T} f_i]/\tau\big)$ where $f_i \in [0,1]$ is a function of state X_i; however we could not use them to prove the concentration hypothesis of Theorem 4. Let us now delve into the proof.

Proof. Observe the random walk performed by FullMassApprox. Clearly if $\bar{\tau} = \Omega(n)$ then the walk visits $O(\tau \ln n + \sqrt{\bar{\tau} n})$ distinct states, and the theorem holds unconditionally. Let us then assume $\bar{\tau} = o(n)$. We disregard the first $T_0 = \Theta(\tau \ln n)$ steps of the walk, which of course yield at most T_0 distinct states, and focus on the last T steps, which we denote by $X_1, \ldots, X_T$ (one may thus plug $T + T_0$ in place of T in the concentration hypothesis). Let π_i denote the distribution of state X_i, $i = 1, \ldots, T$. Since $T_0 = \Theta(\tau \ln n)$, then we can make $\|\pi_i - \pi\|_{\mathrm{TV}} \leq \frac{1}{\operatorname{poly}(n)}$. One can adapt the proof of Lemma 11 to FullMassApprox, using the hypothesis $\|\pi - \pi_i\|_{\mathrm{TV}} \leq \big(\frac{\epsilon \|\pi\|}{\ln(3/\delta)}\big)^c$ for all $i \geq 1$. This changes the bounds of the lemma only by constant multiplicative factors. We can thus focus on proving the bound on the number of states visited by the walk. In the analysis we assume $X_i \sim \pi$, but again the same asymptotic bounds hold if $\|\pi_i - \pi\|_{\mathrm{TV}} \leq \frac{1}{\operatorname{poly}(n)}$. Let $S_{v,t} = \cup_{i=1}^{t} \{X_i\}$, let $N_{v,t} = |S_{v,t}|$, and let $M_{v,t} = \sum_{u \in S_{v,t}} \pi(u)$. For brevity we simply write S_t, N_t, M_t.

The crux is to show that M_t, the aggregate mass of S_t, grows basically as N_t^2/t. Formally we prove that, for any $\epsilon, \delta, q > 0$, if $\Pr[N_t \geq q] \geq 1 - \delta$ then $\Pr[M_t \geq q^2 \frac{\epsilon}{4tn}] \geq 1 - \epsilon - \delta$. First, for any $\lambda > 0$ let $V_\lambda = \{u \in V : \pi(u) < \frac{\lambda}{n}\}$. Clearly $\Pr[X_i \in V_\lambda] = \sum_{u \in V_\lambda} \pi(u) < \lambda$. Therefore the number of steps $J_t(\lambda)$ the chain was on a state of V_λ satisfies $\mathbb{E}[J_t(\lambda)] < t\lambda$. Now, by Markov's inequality $\Pr[J_t(\lambda) > \frac{q}{2}] < \frac{2t\lambda}{q}$, and setting $\lambda = \epsilon \frac{q}{2} t$ we obtain $\Pr[J_t(\lambda) > \frac{q}{2}] < \epsilon$. Since by hypothesis $\Pr[N_t < q] < \delta$, by a union bound we get $\Pr[N_t \geq q,\ J_t(\lambda) < \frac{q}{2}] \geq 1 - \delta - \epsilon$. But if $N_t \geq q$ and $J_t(\lambda) < \frac{q}{2}$ then S_t contains at least $\frac{q}{2}$ distinct states with individual mass at least $\frac{\epsilon q}{2tn}$, and thus $M_t \geq \frac{q}{2} \frac{\epsilon q}{2tn} = q^2 \frac{\epsilon}{4tn}$.

Now choose t such that $\mathbb{E}[N_t] = \Omega(\sqrt{n\bar{\tau}})$; note that $\mathbb{E}[N_t] = \Omega(\bar{\tau})$ since $\bar{\tau} = o(n)$. By plugging $\mathbb{E}[N_t]$ into the concentration bound for N_t we can then make $\Pr[N_t < (1 - \bar{\epsilon})\mathbb{E}[N_t]]$ arbitrarily small for any $\bar{\epsilon} > 0$. Let then $q = (1 - \bar{\epsilon})\mathbb{E}[N_t]$. By the bounds of the previous paragraph, for any $\delta > 0$ with probability $1 - \delta - \bar{\epsilon}$ we have $M_t \geq q^2 \frac{\bar{\epsilon}}{4tn} = \Omega(n\bar{\tau}) \frac{\bar{\epsilon}}{4tn} = \Omega(\frac{\bar{\tau}}{t})$. Conditioned on the event that $M_t = \Omega(\frac{\bar{\tau}}{t})$, any sample drawn after t steps is a repeat with

probability $\Omega(\frac{\bar{\tau}}{t})$. If we then draw $\Theta(\frac{t}{\bar{\tau}})$ samples, which require $\Theta(t)$ steps, we witness an expected $\Omega(1)$ samples, which can be made larger than $k_{\epsilon,\delta}$ by appropriately increasing t. Again by the concentration bounds on N_t, the total number of states visited can be made $O(2\mathbb{E}[Q_t]) = O(\sqrt{n\bar{\tau}}) = \tilde{O}(\sqrt{n\tau})$ with probability arbitrarily close to 1 by appropriately increasing t. ◀

4 Conclusions

We have given improved, optimal algorithms for approximating the stationary probability of a given state in a time-reversible Markov chain, and for approximating the sum of nonnegative real vectors by weighted sampling. Although time-reversible chains are of clear relevance, extending our results to other classes of Markov chains is an intriguing open question. We have also shown that the footprint of our algorithms in terms of number of distinct states visited is tied to the concentration of the number of distinct states visited by the chain; investigating such a concentration is thus an obvious line of future research.

References

1 Noga Alon, Ori Gurel-Gurevich, and Eyal Lubetzky. Choice-memory tradeoff in allocations. *The Annals of Applied Probability*, 20(4):1470–1511, 2010.

2 Anne Auger and Benjamin Doerr, editors. *Theory of Randomized Search Heuristics: Foundations and Recent Developments*, volume 1. World Scientific Publishing Co., Inc., 2011.

3 Siddhartha Banerjee and Peter Lofgren. Fast bidirectional probability estimation in Markov models. In *Proc. of NIPS*, pages 1423–1431, 2015.

4 Phillip Bonacich and Paulette Lloyd. Eigenvector-like measures of centrality for asymmetric relations. *Social Networks*, 23(3):191–201, 2001.

5 Christian Borgs, Michael Brautbar, Jennifer T. Chayes, and Shang-Hua Teng. A sublinear time algorithm for pagerank computations. In Anthony Bonato and Jeannette C. M. Janssen, editors, *Algorithms and Models for the Web Graph - 9th International Workshop, WAW 2012, Halifax, NS, Canada, June 22-23, 2012. Proceedings*, volume 7323 of *Lecture Notes in Computer Science*, pages 41–53. Springer, 2012. doi:10.1007/978-3-642-30541-2_4.

6 Christian Borgs, Michael Brautbar, Jennifer T. Chayes, and Shang-Hua Teng. Multiscale matrix sampling and sublinear-time PageRank computation. *Internet Mathematics*, 10(1-2):20–48, 2014.

7 Marco Bressan, Enoch Peserico, and Luca Pretto. On approximating the stationary distribution of time-reversible Markov chains. *CoRR*, abs/1801.00196, 2018.

8 Kai-Min Chung, Henry Lam, Zhenming Liu, and Michael Mitzenmacher. Chernoff-Hoeffding bounds for Markov chains: Generalized and simplified. In *Proc. of STACS*, pages 124–135, 2012.

9 David A. Freedman. On tail probabilities for martingales. *The Annals of Probability*, 3(1):100–118, 1975.

10 Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Matrix Computations. Johns Hopkins University Press, 2012.

11 Wilfred K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

12 Christina E. Lee, Asuman Ozdaglar, and Devavrat Shah. Computing the stationary distribution, locally. In *Proc. of NIPS*, pages 1376–1384, 2013.

13 Christina E. Lee, Asuman E. Ozdaglar, and Devavrat Shah. Solving systems of linear equations: Locally and asynchronously. *CoRR*, abs/1411.2647, 2014.

14 David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2009.

15 Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. Bidirectional PageRank estimation: from average-case to worst-case. In *Proc. of WAW*, pages 164–176, 2015.

16 Peter A. Lofgren, Siddhartha Banerjee, Ashish Goel, and C. Seshadhri. FAST-PPR: Scaling personalized PageRank estimation for large graphs. In *Proc. of ACM KDD*, pages 1436–1445, 2014.

17 Rajeev Motwani, Rina Panigrahy, and Ying Xu. Estimating sum by weighted sampling. In *Proc. of ICALP*, pages 53–64, 2007.

18 Alessandro Panconesi and Aravind Srinivasan. Randomized distributed edge coloring via an extension of the Chernoff–Hoeffding bounds. *SIAM Journal on Computing*, 26(2):350–368, 1997.

19 Ronitt Rubinfeld and Asaf Shapira. Sublinear time algorithms. *SIAM Journal on Discrete Mathematics*, 25(4):1562–1588, 2011.

20 Nitin Shyamkumar, Siddhartha Banerjee, and Peter Lofgren. Sublinear estimation of a single element in sparse linear systems. In *2016 Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 856–860, 2016.

5 Appendix

5.1 Probability bounds

This appendix provides Chernoff-type probability bounds that are repeatedly used in our analysis; these bounds can be found in e.g. [2], and can be derived from [18].

Let $X_1, \ldots, X_n$ be binary random variables. We say that $X_1, \ldots, X_n$ are non-positively correlated if for all $I \subseteq \{1, \ldots, n\}$ we have:

$$Pr[\forall i \in I : X_i = 0] \leq \prod_{i \in I} Pr[X_i = 0] \tag{10}$$

$$Pr[\forall i \in I : X_i = 1] \leq \prod_{i \in I} Pr[X_i = 1] \tag{11}$$

The following lemma holds:

▶ **Lemma 12.** *Let $X_1, \ldots, X_n$ be independent or, more generally, non-positively correlated binary random variables. Let $a_1, \ldots, a_n \in [0, 1]$ and $X = \sum_{i=1}^{n} a_i X_i$. Then, for any $\epsilon > 0$, we have:*

$$Pr[X < (1 - \epsilon)\mathbb{E}[X]] < e^{-\frac{\epsilon^2}{2}\mathbb{E}[X]} \tag{12}$$

$$Pr[X > (1 + \epsilon)\mathbb{E}[X]] < e^{-\frac{\epsilon^2}{2+\epsilon}\mathbb{E}[X]} \tag{13}$$

Note that Lemma 12 applies if $X_1, \ldots, X_n$ are indicator variables of mutually disjoint events, or if they can be partitioned into independent families $\{X_1, \ldots, X_{i_1}\}$, $\{X_{i_1+1}, \ldots, X_{i_2}\}$, ... of such variables.

5.2 A lower bound for non-time-reversible Markov chains

▶ **Lemma 13.** *For any functions $\tau(n) = \omega(1)$ and $p(n) = o(\frac{1}{n})$ there exists a family of ergodic non-time-reversible Markov chains on n states having mixing time $\tau = \Theta(\tau(n))$, and containing a state v with $\pi(v) = \Theta(p(n))$ such that any algorithm needs $\Omega(\frac{\tau}{\pi(v)})$ calls to step() to estimate $\pi(v)$ within constant multiplicative factors with constant probability.*

Proof. Consider a chain with state space $\{u\} \cup \{u_1, \ldots, u_{n-1}\}$ and the following transition probabilities (we assume n large enough to set in $[0, 1]$ any quantity where needed). For u, set $p_{uu} = 1 - \frac{(n-1)p(n)}{\tau(n)}$, and $p_{uu_i} = \frac{p(n)}{\tau(n)}$ for all $i = 1, \ldots, n-1$. For all $i = 1, \ldots, n-1$, set $p_{u_i u_i} = 1 - \frac{1}{\tau(n)}$ and $p_{u_i u} = \frac{1}{\tau(n)}$. The chain is clearly ergodic. Note that $\frac{(n-1)p(n)}{\tau(n)} = o(\frac{1}{\tau(n)})$ and therefore the expected time to leave u is asymptotically larger than the expected time to leave any of the u_i. One can then check that (i) $\pi(u_i) = \Theta(p(n))$, and (ii) the mixing time is $\tau = \Theta(\tau(n))$ (essentially, the expected time to leave the u_i). Pick any u_i as target state v. Suppose now to alter the chain as follows: pick some $u_j \neq v$ and set $p_{u_j v} = 1$. The new stationary probability of v would then be roughly $2\pi(v)$. However one cannot distinguish between the two chains with constant probability with less than $\Omega(\frac{\tau}{\pi(v)})$ *step*() calls. Indeed, to distinguish between them one must at least visit u_j (and then perform e.g. *probe*(u_j, v)). Since u is the only state leading to u_j with positive probability, one must invoke *step*(u) until it returns u_j. But $p_{uu_j} = \frac{p(n)}{\tau(n)}$, hence one needs $\Omega(\frac{\tau(n)}{p(n)}) = \Omega(\frac{\tau}{\pi(v)})$ calls in expectation. The construction can be adapted to any constant approximation factor by adding more transitions towards v. ◀

On Singleton Arc Consistency for CSPs Defined by Monotone Patterns

Clément Carbonnel
Department of Computer Science, University of Oxford, UK
clement.carbonnel@cs.ox.ac.uk

David A. Cohen
Royal Holloway, University of London, UK
d.cohen@rhul.ac.uk

Martin C. Cooper
IRIT, University of Toulouse III, France
cooper@irit.fr

Stanislav Živný
Department of Computer Science, University of Oxford, UK
standa.zivny@cs.ox.ac.uk

Abstract

Singleton arc consistency is an important type of local consistency which has been recently shown to solve all constraint satisfaction problems (CSPs) over constraint languages of bounded width. We aim to characterise all classes of CSPs defined by a forbidden pattern that are solved by singleton arc consistency and closed under removing constraints. We identify five new patterns whose absence ensures solvability by singleton arc consistency, four of which are provably maximal and three of which generalise 2-SAT. Combined with simple counter-examples for other patterns, we make significant progress towards a complete classification.

2012 ACM Subject Classification Theory of computation → Complexity classes, Theory of computation → Problems, reductions and completeness, Mathematics of computing → Discrete mathematics

Keywords and phrases Constraint Satisfaction Problems, Forbidden Patterns, Singleton Arc Consistency

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.19

Related Version A full version of the paper is available at `http://arxiv.org/abs/1704.06215`, [12].

Funding The authors were supported by EPSRC grant EP/L021226/1 and ANR-11-LABX-0040-CIMI within the program ANR-11-IDEX-0002-02. Stanislav Živný was supported by a Royal Society University Research Fellowship. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 714532). The paper reflects only the authors' views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 19; pp. 19:1–19:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The constraint satisfaction problem (CSP) is a declarative paradigm for expressing computational problems. An instance of the CSP consists of a number of variables to which we need to assign values from some domain. Some subsets of the variables are constrained in that they are not permitted to take all values in the product of their domains. The scope of a constraint is the set of variables whose values are limited by the constraint, and the constraint relation is the set of permitted assignments to the variables of the scope. A solution to a CSP instance is an assignment of values to variables in such a way that every constraint is satisfied, i.e. every scope is assigned to an element of the constraint relation.

The CSP has proved to be a useful technique for modelling in many important application areas from manufacturing to process optimisation, for example planning and scheduling optimisation [31], resource allocation [29], job shop problems [14] and workflow management [32]. Hence much work has been done on describing useful classes of constraints [3] and implementing efficient algorithms for processing constraints [7]. Many constraint solvers use a form of backtracking where successive variables are assigned values that satisfy all constraints. In order to mitigate the exponential complexity of backtracking some form of pre-processing is always performed. These pre-processing techniques identify values that cannot be part of any solution in an effective way and then propagate the effects of removing these values throughout the problem instance. Of key importance amongst these pre-processing algorithms are the relatives of arc consistency propagation including generalised arc consistency (GAC) and singleton arc consistency (SAC). Surprisingly there are large classes [17, 23, 13, 28] of the CSP for which GAC or SAC are decision procedures: after establishing consistency if every variable still has a non-empty domain then the instance has a solution.

More generally, these results fit into the wider area of research aiming to identify subproblems of the CSP for which certain polynomial-time algorithms are decision procedures. Perhaps the most natural ways to restrict the CSP is to limit the constraint relations that we allow or to limit the structure of (the hypergraph of) interactions of the constraint scopes. A set of allowed constraint relations is called a constraint language. A subset of the CSP defined by limiting the scope interactions is called a structural class.

There has been considerable success in identifying tractable constraint languages, recently yielding a full classification of the complexity of finite constraint languages [9, 33]. Techniques from universal algebra have been essential in this work as the complexity of a constraint language is characterised by a particular algebraic structure [11]. The two most important algorithms for solving the CSP over tractable constraint languages are local consistency and the few subpowers algorithm [10, 27], which generalises ideas from group theory. A necessary and sufficient condition for solvability by the few subpowers algorithm was identified in [27, 4]. The set of all constraint languages decided by local consistency was later described by Barto and Kozik [2] and independently by Bulatov [8]. Surprisingly, all such languages are in fact decided by establishing singleton arc consistency [28].

A necessary condition for the tractability of a structural class with bounded arity is that it has bounded treewidth modulo homomorphic equivalence [26]. In all such cases we decide an instance by establishing k-consistency, where k is the treewidth of the core. It was later shown that the converse holds: if a class of structures does not have treewidth k modulo homomorphic equivalence then it is not solved by k-consistency [1], thus fully characterising the strength of consistency algorithms for structural restrictions. Both language-restricted CSPs and CSPs of bounded treewidth are *monotone* in the sense that we can relax (remove constraints from) any CSP instance without affecting its membership in such a class.

Since our understanding of consistency algorithms for language and structural classes is so well advanced there is now much interest in so called hybrid classes, which are neither definable by restricting the language nor by limiting the structure. For the binary CSP, one popular mechanism for defining hybrid classes follows the considerable success of mapping the complexity landscape for graph problems in the absence of certain induced subgraphs or graph minors. Here, hybrid (binary) CSP problems are defined by forbidding a fixed set of substructures (*patterns*) from occurring in the instance [15]. This framework is particularly useful in algorithm analysis, since it allows us to identify precisely the local properties of a CSP instance that make it impossible to solve via a given polynomial-time algorithm. This approach has recently been used to obtain a pattern-based characterisation of solvability by arc consistency [23], a detailed analysis of variable elimination rules [16] and various novel tractable classes of CSP [20, 19].

Singleton arc consistency is a prime candidate to study in this framework since it is one of the most prominent incomplete polynomial-time algorithms for CSP and the highest level of consistency (among commonly studied consistencies) that operates only by removing values from domains. This property ensures that enforcing SAC cannot introduce new patterns, which greatly facilitates the analysis. It is therefore natural to ask for which patterns, forbidding their occurrence ensures that SAC is a sound decision procedure. In this paper we make a significant contribution towards this objective by identifying five patterns which define classes of CSPs decidable by SAC. All five classes are monotone, and we show that only a handful of open cases separates us from an essentially full characterisation of monotone CSP classes decidable by SAC and definable by a forbidden pattern. Some of our results rely on a novel proof technique which follows the *trace* of a successful run of the SAC algorithm to dynamically identify redundant substructures in the instance and construct a solution.

The structure of the paper is as follows. In Section 2 we provide essential definitions and background theory. In Section 3 we state the main results. The rest of the paper includes some of the proofs. All remaining proofs are provided in the long version [12].

2 Preliminaries

CSP. A *binary CSP instance* is a triple $I = (X, D, C)$, where X is a finite set of variables, D is a finite domain, each variable $x \in X$ has its own domain of possible values $D(x) \subseteq D$, and $C = \{R(x, y) \mid x, y \in X, x \neq y\}$, where $R(x, y) \subseteq D^2$, is the set of constraints. We assume, without loss of generality, that each pair of variables $x, y \in X$ is constrained by a constraint $R(x, y)$. (Otherwise we set $R(x, y) = D(x) \times D(y)$.) We also assume that $(a, b) \in R(x, y)$ if and only if $(b, a) \in R(y, x)$. A constraint is *trivial* if it contains the Cartesian product of the domains of the two variables. By *deleting* a constraint we mean replacing it with a trivial constraint. The *projection* $I[X']$ of a binary CSP instance I on $X' \subseteq X$ is obtained by removing all variables in $X \backslash X'$ and all constraints $R(x, y)$ with $\{x, y\} \not\subseteq X'$. A *partial solution* to a binary CSP instance on $X' \subseteq X$ is an assignment s of values to variables in X' such that $s(x) \in D(x)$ for all $x \in X'$ and $(s(x), s(y)) \in R(x, y)$ for all constraints $R(x, y)$ with $x, y \in X'$. A *solution* to a binary CSP instance is a partial solution on X.

An assignment (x, a) is called a *point*. If $(a, b) \in R(x, y)$, we say that the assignments $(x, a), (y, b)$ (or more simply a, b) are *compatible* and that ab is a *positive edge*, otherwise a, b are *incompatible* and ab is a *negative edge*. For simplicity of notation we can assume that variable domains are disjoint, so that using a as a shorthand for (x, a) is unambiguous. We say that $a \in D(x)$ has a *support* at variable y if $\exists b \in D(y)$ such that ab is a positive edge.

The constraint graph of a CSP instance with variables X is the graph $G = (X, E)$ such that $(x, y) \in E$ if $R(x, y)$ is non-trivial. The *degree* of a variable x in a CSP instance is the degree of x in the constraint graph of the instance.

Arc Consistency. A domain value $a \in D(x)$ is *arc consistent* if it has a support at every other variable. A CSP instance is *arc consistent* (AC) if every domain value is arc consistent.

Singleton Arc Consistency. Singleton arc consistency is stronger than arc consistency (but weaker than strong path consistency [30]). A domain value $a \in D(x)$ in a CSP instance I is *singleton arc consistent* if the instance obtained from I by removing all domain values $b \in D(x)$ with $a \neq b$ can be made arc consistent without emptying any domain. A CSP instance is *singleton arc consistent* (SAC) if every domain value is singleton arc consistent.

Establishing Consistency. Domain values that are not arc consistent or not singleton arc consistent cannot be part of a solution so can safely be removed. For a binary CSP instance with domain size d, n variables and e non-trivial constraints there are $O(ed^2)$ algorithms for establishing arc consistency [6] and $O(ned^3)$ algorithms for establishing singleton arc consistency [5]. These algorithms repeatedly remove inconsistent values from domains.

SAC *decides* a CSP instance if, after establishing singleton arc consistency, non-empty domains for all variables guarantee the existence of a solution. SAC decides a class of CSP instances if SAC decides every instance from the class.

Neighbourhood Substitutability. If $a, b \in D(x)$, then a is *neighbourhood substitutable* by b if there is no c such that ac is a positive edge and bc a negative edge: such values a can be deleted from $D(x)$ without changing the satisfiability of the instance since a can be replaced by b in any solution [25]. Similarly, removing neighbourhood substitutable values cannot destroy (singleton) arc consistency.

Patterns. In a binary CSP instance each constraint decides, for each pair of values in D, whether it is allowed. Hence a binary CSP can also be defined as a set of points $X \times D$ together with a compatibility function that maps each edge, $((x, a), (y, b))$ with $x \neq y$, into the set $\{\text{negative}, \text{positive}\}$. A *pattern* extends the notion of a binary CSP instance by allowing the compatibility function to be partial. A pattern P *occurs* (as a subpattern) in an instance I if there is mapping from the points of P to the points of I which respects variables (two points are mapped to points of the same variable in I if and only if they belong to the same variable in P) and maps negative edges to negative edges, and positive edges to positive edges. A set of patterns occurs in an instance I if at least one pattern in the set occurs in I.

We use the notation $\text{CSP}(\overline{P})$ for the set of binary instances in which P does not occur as a subpattern. A pattern P is *SAC-solvable* if SAC decides $\text{CSP}(\overline{P})$. It is worth observing that $\text{CSP}(\overline{P})$ is closed under the operation of establishing (singleton) arc consistency. A pattern P is *tractable* if $\text{CSP}(\overline{P})$ can be solved in polynomial time.

Points (x, a) and (x, b) in a pattern are *mergeable* if there is no point (y, c) such that ac is positive and bc is negative or vice versa. For each set of patterns there is an equivalent set of patterns without mergeable points which occur in the same set of instances.

A point (x, a) in a pattern is called *dangling* if there is at most one b such that ab is a positive edge and no c such that ac is a negative edge. Dangling points are redundant when considering the occurrence of a pattern in an arc consistent CSP instance.

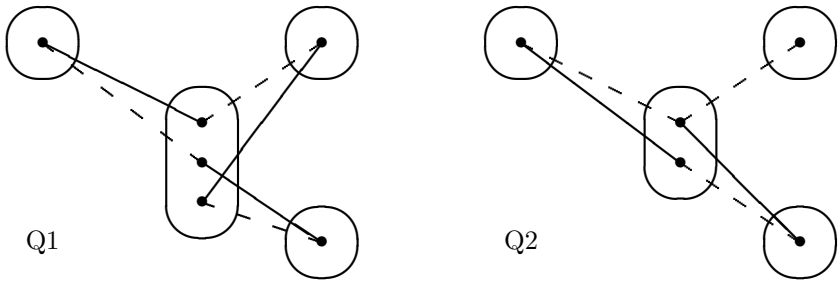


Figure 1 All degree-3 irreducible monotone patterns solved by SAC must occur in at least one of these patterns.

A pattern is called *irreducible* if it has no dangling points and no mergeable points [20]. When studying algorithms that are at least as strong as arc consistency, a classification with respect to forbidden *sets* of irreducible patterns is equivalent to a classification with respect to all forbidden sets of patterns. For this reason classifications are often established with respect to irreducible patterns even if only classes definable by forbidding a single pattern are considered [20, 23], as we do in the present paper.

3 Results

Call a class $\mathcal{C}$ of CSP instances *monotone* if deleting any constraint from an instance $I \in \mathcal{C}$ produces another instance in $\mathcal{C}$. For example, language classes and bounded treewidth classes are monotone. An interesting research direction is to study those monotone classes defined by a forbidden pattern which are solved by singleton arc consistency, both in order to uncover new tractable classes and to better understand the strength of SAC.

We call a pattern *monotone* if when forbidden it defines a monotone class. Monotone patterns can easily be seen to correspond to exactly those patterns in which positive edges only occur in constraints which have at least one negative edge.

Consider the monotone patterns Q1 and Q2 shown in Figure 1, patterns R5, R8 shown in Figure 2, and pattern R7- shown in Figure 3.

▶ **Theorem** (Main). *The patterns Q1, Q2, R5, R8, and R7- are SAC-solvable.*

In order to prove the SAC-solvability of Q1, R8 and R7- we use the same idea of following the trace of arc consistency and argue that the resulting instance is not too complicated. While the same idea is behind the proofs of all three patterns, the technical details differ.

In the remaining two cases we identify an operation that preserves SAC and satisfiability, does not introduce the pattern and after repeated application necessarily produces an equivalent instance which is solved by SAC. In the case of R5, the operation is simply removing any constraint. In the case of Q2, the operation is BTP-merging [19].

▶ Remark. The full version of this paper [12] tells us that any *monotone* and *irreducible* pattern solvable by SAC must occur in at least one of the patterns shown in Figures 1 and 2. By this analysis, we have managed to reduce the number of remaining cases to a handful. Our main result shows that some of these are SAC-solvable. In particular, the patterns Q1, Q2, R5, and R8 are maximal in the sense that adding anything to them would give a pattern that is either non-monotone or not solved by SAC.

▶ Remark. We point out that certain interesting forbidden patterns, such as BTP [21], NegTrans [22], and EMC [23] are not monotone. On the other hand, the patterns T1, . . . ,T5

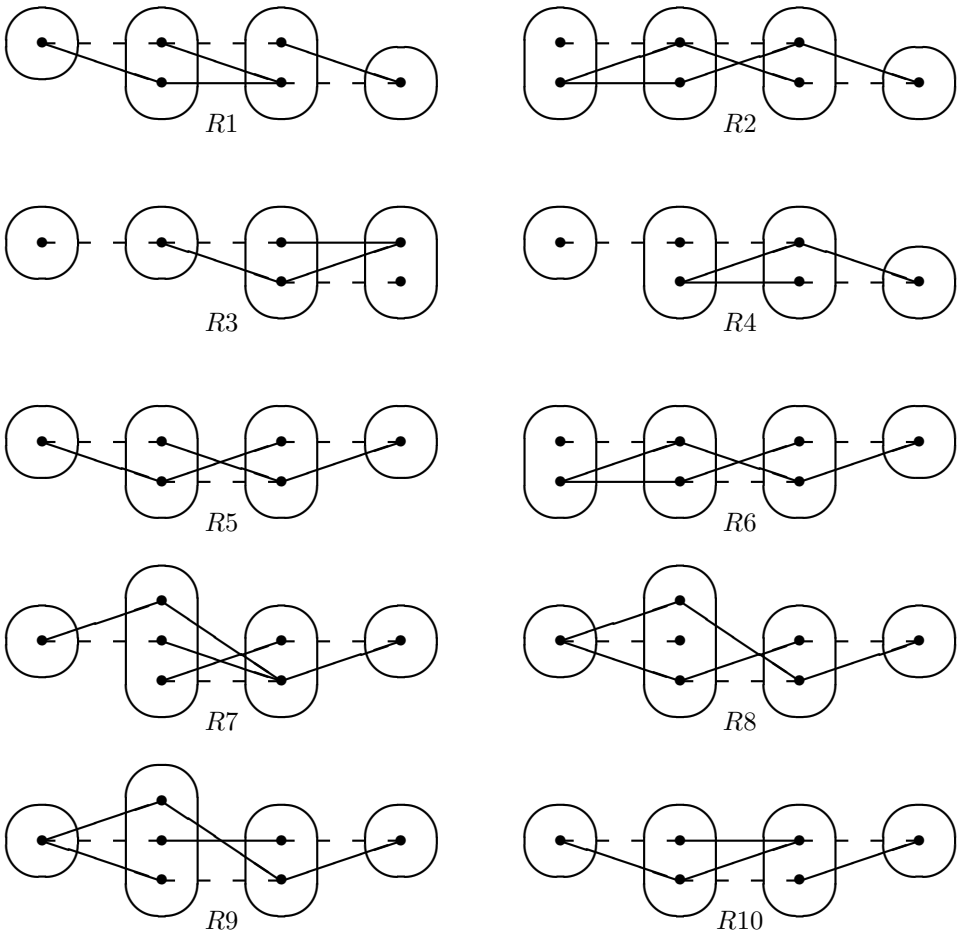


Figure 2 All degree-2 irreducible monotone patterns solved by SAC must occur in at least one of these patterns.

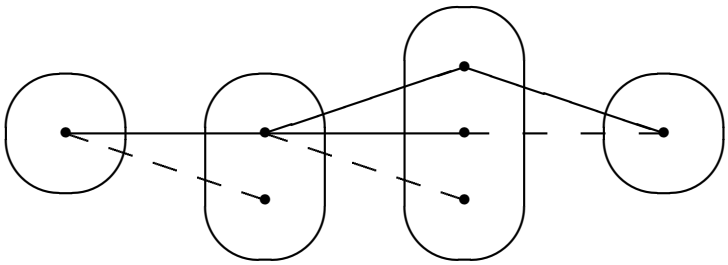


Figure 3 The pattern R7-, a subpattern of R7.

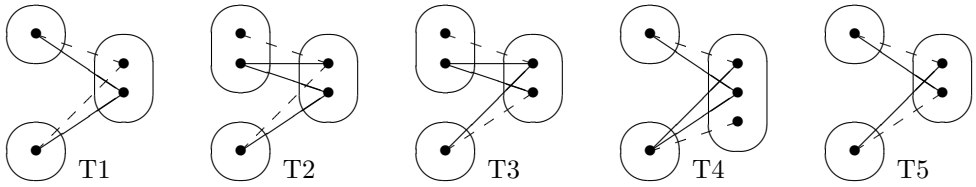


Figure 4 The set of tractable 2-constraint irreducible patterns.

shown in Figure 4 *are* monotone. Patterns T1,...,T5 were identified in [20] as the maximal irreducible tractable patterns on two connected constraints. We show in [12] that T1 is not solved by SAC. Our main result implies (since R8 contains T4 and T5) that both T4 and T5 are solved by SAC. It can easily be shown, from Lemma 12 and [20, Lemma 25], that T2 is solved by SAC, and we provide in [12] a simple proof that T3 is solved by SAC as well. Hence, we have characterised all 2-constraint irreducible patterns solvable by SAC.

▶ Remark. Observe that Q1 does not occur in any binary CSP instance in which all degree 3 or more variables are Boolean. This shows that 2-SAT is a strict subset of CSP($\overline{\text{Q1}}$). This class is incomparable with language-based generalisations of 2-SAT, such as the class ZOA [18], since in CSP($\overline{\text{Q1}}$) degree-2 variables can be constrained by arbitrary constraints. Indeed, instances in CSP($\overline{\text{Q1}}$) can have an arbitrary constraint on the pair of variables x, y, where x is of arbitrary degree and of arbitrary domain size if for all variables $z \notin \{x, y\}$, the constraint on the pair of variables x, z is of the form $(x \in S) \vee (z \in T_z)$ where S is fixed (i.e. independent of z) but T_z is arbitrary. R8 and R7- generalise T4 and CSP($\overline{\text{T4}}$) generalises ZOA [20], so CSP($\overline{\text{R8}}$) and CSP($\overline{\text{R7-}}$) are strict generalisations of ZOA.

4 Notation for the Trace Technique

Given a singleton arc consistent instance I, a variable x and a value $v \in D(x)$, we denote by I_{xv} the instance obtained by assigning x to v (that is, setting $D(x) = \{v\}$) and enforcing arc consistency. To avoid confusion with the original domains, we will use $D_{xv}(y)$ to denote the domain of the variable y in I_{xv}. For our proofs we will assume that arc consistency has been enforced using a straightforward algorithm that examines the constraints one at a time and removes the points that do not have a support until a fixed point is reached. We will be interested in the *trace* of this algorithm, given as a chain of propagations:

$$(P_{xv}) : (x \to y_0), (x_1 \to y_1), (x_2 \to y_2), \ldots, (x_p \to y_p)$$

where $x_i \to y_i$ means that the algorithm has inferred a change in the domain of y_i when examining the constraint $R(x_i, y_i)$. We define a map $\rho : (P_{xv}) \mapsto 2^D$ that maps each $(x_i \to y_i) \in (P_{xv})$ to the set of values that were removed from $D_{xv}(y_i)$ at this step. Without loss of generality, we assume that the steps $(x_i \to y_i)$ such that the pruning of $\rho(x_i \to y_i)$ from $D_{xv}(y_i)$ does not incur further propagation are performed last.

We denote by $S_{(P_{xv})}$ the set of variables that appear in (P_{xv}). Because I was (singleton) arc consistent before x was assigned, we have $S_{(P_{xv})} = \{x\} \cup \{y_i \mid i \geq 0\}$. We rename the elements of $S_{(P_{xv})}$ as $\{p_i \mid i \geq 0\}$ where the index i denotes the order of first appearance in (P_{xv}). Finally, we use $S^I_{(P_{xv})}$ to denote the set of *inner* variables, that is, the set of all variables $p_j \in S_{(P_{xv})}$ for which there exists $p_r \in S_{(P_{xv})}$ such that $(p_j \to p_r) \in (P_{xv})$.

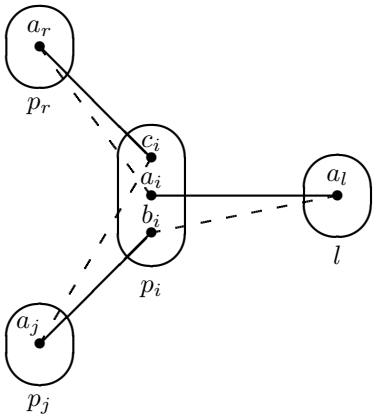


Figure 5 The occurence of Q1 in the proof of Lemma 1.

5 Tractability of Q1

Consider the pattern Q1 shown in Figure 1. Let $I \in \mathrm{CSP}(\overline{\mathrm{Q1}})$ be a singleton arc consistent instance, x be any variable and v be any value in the domain of x. Our proof of the SAC-decidability of $\mathrm{CSP}(\overline{\mathrm{Q1}})$ uses the trace of the arc consistency algorithm to determine a subset of variables in the vicinity of x such that (i) the projection of I_{xv} to this particular subset is satisfiable, (ii) those variables do not interact too much with the rest of the instance and (iii) the projections of I_{xv} and I on the other variables are almost the same. We then use these three properties to show that the satisfiability of I is equivalent to that of an instance with fewer variables, and we repeat the operation until the smaller instance is trivially satisfiable.

The following lemma describes the particular structure of I_{xv} around the variables whose domain has been reduced by arc consistency. Note that a non-trivial constraint in I can be trivial in I_{xv} because of domain changes.

▶ **Lemma 1.** *Consider the instance I_{xv}. Every variable $p_i \in S^I_{(P_{xv})}$ is in the scope of at most two non-trivial constraints, which must be of the form $R(p_j, p_i)$ and $R(p_i, p_r)$ with $j < i$, $(p_j \to p_i) \in (P_{xv})$ and $(p_i \to p_r) \in (P_{xv})$.*

Proof. The claim is true for $p_0 = x$ as every constraint incident to x is trivial. Otherwise, let $p_i \in S^I_{(P_{xv})}$ be such that $p_i \neq x$. Let p_j, $j < i$ be such that $(p_j \to p_i)$ occurs first in (P_{xv}). Because $p_i \in S^I_{(P_{xv})}$ and we assumed that the arc consistency algorithm performs the pruning that do not incur further propagation last, we know that there exists $c_i \in \rho(p_j \to p_i)$ and $p_r \in S_{(P_{xv})}$ with $(p_i \to p_r) \in (P_{xv})$ such that the pruning of c_i from $D_{xv}(p_i)$ allows the pruning of some $a_r \in \rho(p_i \to p_r)$ from the domain of p_r. It follows that $(c_i, a_r) \in R(p_i, p_r)$, $(v_i, a_r) \notin R(p_i, p_r)$ for any $v_i \in D_{xv}(p_i)$ and $(v_j, c_i) \notin R(p_j, p_i)$ for any $v_j \in D_{xv}(p_j)$. Moreover, a_r was a support for c_i at p_r when c_i was pruned so we know that $p_j \neq p_r$.

For the sake of contradiction, let us assume that there exists a constraint $R(p_i, l)$ with $l \notin \{p_j, p_r\}$ that is not trivial. In particular, there exist $a_i, b_i \in D_{xv}(p_i)$ and $a_l \in D_{xv}(l)$ such that $(a_i, a_l) \in R(p_i, l)$ but $(b_i, a_l) \notin R(p_i, l)$. Furthermore, a_r was removed by arc consistency when inspecting the constraint $R(p_i, p_r)$ so $(a_i, a_r) \notin R(p_i, p_r)$. I_{xv} is arc consistent so there exists some $a_j \in D_{xv}(p_j)$ such that $(a_j, b_i) \in R(p_j, p_i)$, and since $c_i \in \rho(p_j \to p_i)$ we have $(a_j, c_i) \notin R(p_j, p_i)$. At this point we have reached the desired contradiction as Q1 occurs on (p_i, p_j, p_r, l) with p_i being the middle variable (see Figure 5). ◀

Given a subset S of variables, an *S-path* between two variables y_1 and y_2 is a path $R(y_1, x_2), R(x_2, x_3), \ldots, R(x_k, y_2)$ of non-trivial constraints with $k \geq 2$ and $x_2, \ldots, x_k \in S$.

▶ **Lemma 2.** *Consider the instance I_{xv}. There is no $(S^I_{(P_{xv})})$-path between two variables in $X\backslash S^I_{(P_{xv})}$ and there is no cycle of non-trivial constraints in $I_{xv}[S^I_{(P_{xv})}]$.*

Proof. Let $y_1, y_2 \in X\backslash S^I_{(P_{xv})}$ and assume for the sake of contradiction that a $(S^I_{(P_{xv})})$-path $R(y_1, x_2), R(x_2, x_3), \ldots, R(x_{k-1}, y_2)$ exists. Let $p_i \in \{x_2, \ldots, x_{k-1}\}$ be such that i is minimum. Since p_i is in the scope of two non-trivial constraints in this path, it follows from Lemma 1 that p_i is in the scope of exactly two non-trivial constraints, one of which is of the form $R(p_j, p_i)$ with $j < i$ and $(p_j \to p_i) \in (P_{xv})$. It follows from $(p_j \to p_i) \in (P_{xv})$ that $p_j \in S^I_{(P_{xv})}$ and hence p_j is not an endpoint of the path, and then $j < i$ contradicts the minimality of i. The second part of the claim follows from the same argument, by considering a cycle as a $(S^I_{(P_{xv})})$-path $R(x_1, x_2), R(x_2, x_3), \ldots, R(x_{k-1}, x_1)$ with $x_1 \in (S^I_{(P_{xv})})$ and defining p_i as the variable among $\{x_1, \ldots, x_{k-1}\}$ with minimum index. ◀

▶ **Lemma 3.** *I_{xv} has a solution if and only if $I_{xv}[X\backslash S^I_{(P_{xv})}]$ has a solution.*

Proof. The "only if" implication is trivial, so we focus on the other direction. Suppose that there exists a solution ϕ to $I_{xv}[X\backslash S^I_{(P_{xv})}]$. Let Y be a set of variables initialized to $X\backslash S^I_{(P_{xv})}$. We will grow Y with the invariants that (i) we know a solution ϕ to $I_{xv}[Y]$, and (ii) there is no $(X\backslash Y)$-path between two variables in Y (which is true at the initial state by Lemma 2).

If there is no non-trivial constraint between $X\backslash Y$ and Y then I_{xv} is satisfiable if and only if $I_{xv}[X\backslash Y]$ is. By construction $X\backslash Y \subseteq S^I_{(P_{xv})}$ and by Lemma 2 we know that $I_{xv}[X\backslash Y]$ has no cycle of non-trivial constraints. Because $I_{xv}[X\backslash Y]$ is arc consistent and acyclic it has a solution [24], and we can conclude that in this case I_{xv} has a solution.

Otherwise, let $p_i \in X\backslash Y$ be such that there exists a non-trivial constraint between p_i and some variable $p_r \in Y$. By (ii), this non-trivial constraint must be unique (with respect to p_i) as otherwise we would have a $(X\backslash Y)$-path between two variables in Y. By arc consistency, there exists $a_i \in D_{xv}(p_i)$ such that $(a_i, \phi(p_r)) \in R(p_i, p_r)$; because this non-trivial constraint is unique, setting $\phi(p_i) = a_i$ yields a solution to $I_{xv}[Y \cup \{p_i\}]$. Because any $(X\backslash(Y \cup \{p_i\}))$-path between two variables in $Y \cup \{p_i\}$ would extend to a $(X\backslash Y)$-path between Y variables by going through p_i, we know that no such path exists. Then $Y \leftarrow Y \cup \{p_i\}$ satisfies both invariants, so we can repeat the operation until we have a solution to the whole instance or all constraints between Y and $X\backslash Y$ are trivial. In both cases I_{xv} has a solution. ◀

▶ **Lemma 4.** *I has a solution if and only if $I[X\backslash S^I_{(P_{xv})}]$ has a solution.*

Proof. Again the "only if" implication is trivial so we focus on the other direction. Let us assume for the sake of contradiction that $I[X\backslash S^I_{(P_{xv})}]$ has a solution but I does not. In particular this implies that I_{xv} does not have a solution, and then by Lemma 3 we know that $I_{xv}[X\backslash S^I_{(P_{xv})}]$ has no solution either. We define Z as a subset of $X\backslash S^I_{(P_{xv})}$ of minimum size such that $I_{xv}[Z]$ has no solution. Observe that $I_{xv}[Z]$ can only differ from $I[Z]$ by having fewer values in the domain of the variables in $S_{(P_{xv})}$. Let ϕ be a solution to $I[Z]$ such that $\phi(y) \in D_{xv}(y)$ for as many variables y as possible. Because ϕ is not a solution to $I_{xv}[Z]$, there exists $p_r \in Z \cap S_{(P_{xv})}$ and $p_j \in S^I_{(P_{xv})}$ such that $(p_j \to p_r) \in (P_{xv})$ and $\phi(p_r) \in \rho(p_j \to p_r)$ (recall that $\rho(p_j \to p_r)$ is the set of points removed by the AC algorithm in the domain of p_r at step $(p_j \to p_r)$). By construction, $p_j \notin Z$.

First, let us assume that there exists a variable $y \in Z$, $y \neq p_r$ such that there is no $a_r \in D_{xv}(p_r)$ with $(\phi(y), a_r) \in R(y, p_r)$. This implies, in particular, that $\phi(y) \notin D_{xv}(y)$. We first prove that $R(y, p_r)$ and $R(p_j, p_r)$ are the only possible non-trivial constraints involving p_r in I_{xv}. If there exists a fourth variable z such that $R(p_r, z)$ is non-trivial in I_{xv}, then there exist $a_r, b_r \in D_{xv}(p_r)$ and $a_z \in D_{xv}(z)$ such that $(a_r, a_z) \in R(p_r, z)$ but $(b_r, a_z) \notin R(p_r, z)$.

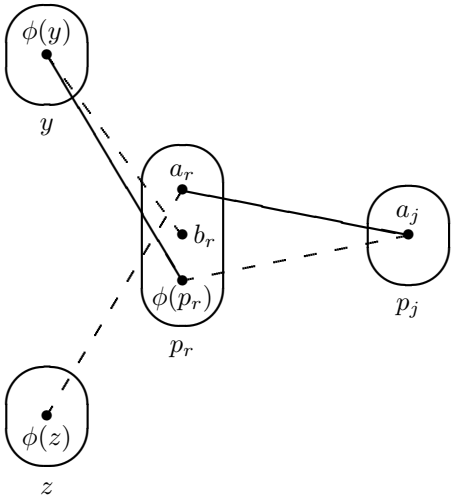


Figure 6 Some positive and negative edges between y, z, p_j and p_r. The positive edges $(\phi(y), a_r)$ and $(\phi(z), \phi(p_r))$ are omitted for clarity; b_r is any value in $D_{xv}(p_r)$ that is not compatible with $\phi(y)$.

By assumption we have $(\phi(y), a_r) \notin R(y, p_r)$ and $(\phi(y), \phi(p_r)) \in R(y, p_r)$. Finally, b_r has a support $a_j \in D_{xv}(p_j)$ and $\phi(p_r) \in \rho(p_j \to p_r)$ so we have $(a_j, a_r) \in R(p_j, p_r)$ but $(a_j, \phi(p_r)) \notin R(p_j, p_r)$. This produces Q1 on (p_r, y, p_j, z) with p_r being the middle variable. Therefore, we know that $R(y, p_r)$ and $R(p_j, p_r)$ are the only possible non-trivial constraints involving p_r in I_{xv}. However, in this case the variable p_r has only one incident non-trivial constraint in $I_{xv}[Z]$, and hence $I_{xv}[Z]$ has a solution if and only if $I_{xv}[Z \backslash p_r]$ has one. This contradicts the minimality of Z, and for the rest of the proof we can assume that for every $y \in Z$ there exists some $a_r \neq \phi(p_r)$ such that $a_r \in D_{xv}(p_r)$ and $(\phi(y), a_r) \in R(y, p_r)$.

Now, let $y \in Z$ be such that $y \neq p_r$ and $|\{b \in D_{xv}(p_r) \mid (\phi(y), b) \in R(y, p_r)\}|$ is minimum. By the argument above, there exists $a_r \in D_{xv}(p_r)$ such that $(\phi(y), a_r) \in R(y, p_r)$ and $a_r \neq \phi(p_r)$. By hypothesis setting $\phi(p_r) = a_r$ would violate at least one constraint in $I[Z]$, so there exists some variable $z \in Z$, $z \neq y$ such that $(\phi(z), a_r) \notin R(z, p_r)$. Furthermore, by arc consistency of I_{xv} there exists $a_j \in D_{xv}(p_j)$ such that $(a_j, a_r) \in R(p_j, p_r)$. Recall that we picked p_j in such a way that $\phi(p_r) \in \rho(p_j \to p_r)$, and so we have $(a_j, \phi(p_r)) \notin R(p_j, p_r)$. We summarize what we have in Figure 6. Observe that unless Q1 occurs, for every $b_r \in D_{xv}(p_r)$ such that $(\phi(y), b_r) \notin R(y, p_r)$ we also have $(\phi(z), b_r) \notin R(z, p_r)$. However, recall that $(\phi(y), a_r) \in R(y, p_r)$ so $\phi(z)$ is compatible with strictly fewer values in $D_{xv}(p_r)$ than $\phi(y)$. This contradicts the choice of y. It follows that setting $\phi(p_r) = a_r$ cannot violate any constraint in $I[Z]$, which is impossible by our choice of ϕ - a final contradiction. ◀

▶ **Theorem 5.** $\mathrm{CSP}(\overline{Q1})$ *is solved by singleton arc consistency.*

Proof. Let $I \in \mathrm{CSP}(\overline{\mathrm{Q1}})$ be singleton arc consistent. Pick any variable x and value $v \in D(x)$. By singleton arc consistency the instance I_{xv} does not have any empty domains. By Lemma 4, I has a solution if and only if $I[X \backslash S^I_{(P_{xv})}]$ has one. Because $I[X \backslash S^I_{(P_{xv})}]$ is singleton arc consistent as well and $S^I_{(P_{xv})} \neq \emptyset$ we can repeat the procedure until $X \backslash S^I_{(P_{xv})}$ is empty, at which point we may conclude that I has a solution. ◀

6 Tractability of R8 and R7-

Q1 and R8 (Figure 2) are structurally dissimilar, but the idea of using I_{xv} and the trace of the arc consistency algorithm to extract variables from I without altering satisfiability works in the case of R8 as well. We define a *star* to be a non-empty set of constraints whose scopes

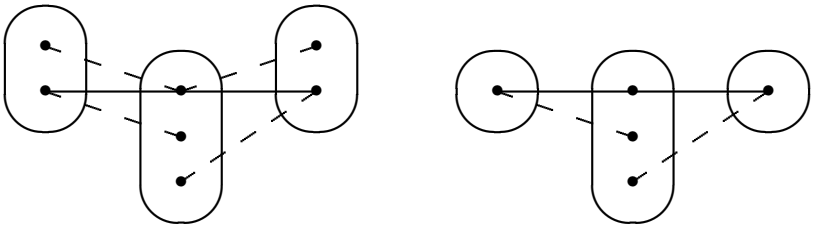


Figure 7 The patterns $\hat{\mathrm{M}}$ (left) and V_2 (right).

all intersect. The *centers* of a star are its variables of highest degree (every star with three or more variables has a unique center). The following lemma is the R8 analog of Lemma 1; the main differences are a slightly stronger prerequisite (no neighbourhood substitutable values) and that arc consistency leaves stars of non-trivial constraints instead of paths.

▶ **Lemma 6.** *Let $I = (X, D, C) \in \mathrm{CSP}(\overline{R8})$ be singleton arc consistent. Let $x \in X$, $v \in D(x)$ and consider the instance I_{xv}. After the removal of every neighbourhood substitutable value, every connected component of non-trivial constraints that intersect with $S_{(P_{xv})}$ is a star with a center in $S_{(P_{xv})}$.*

In the proof of SAC-solvability of Q1, only inner variables are extracted from the instance. The above lemma suggests that in the case of R8 it is more convenient to extract all variables in $S_{(P_{xv})}$, plus any variable that can be reached from those via a non-trivial constraint.

▶ **Lemma 7.** *Let $I = (X, D, C) \in \mathrm{CSP}(\overline{R8})$ be singleton arc consistent. Let $x \in X$, $v \in D(x)$ and consider the instance I_{xv}. There exists a partition (X_1, X_2) of X such that*

- *$S_{(P_{xv})} \subseteq X_1$;*
- *$\forall (x, y) \in X_1 \times X_2$, $R(x, y)$ is trivial;*
- *Every connected component of non-trivial constraints with scopes subsets of X_1 is a star.*

▶ **Theorem 8.** *$\mathrm{CSP}(\overline{R8})$ is solved by singleton arc consistency.*

Our proof of the SAC-solvability of R7- (Figure 3) follows a similar reasoning, with two main differences. First, branching on just any variable-value pair (as we did for Q1 and R8) may lead to a subproblem that is *not* solved by arc consistency. However, once the right assignment is made the reward is much greater as all constraints involving a variable whose domain has been reduced by arc consistency must become trivial *except at most one*.

Finding out which variable we should branch on is tricky. Our proof works by induction, and the ideal starting point is a substructure corresponding to a particular pattern $\hat{\mathrm{M}}$ (Figure 7). However, $\hat{\mathrm{M}}$ is an NP-hard pattern [15] so it may not occur at all in the instance. To handle this problem we define a weaker pattern V_2 (Figure 7), whose absence implies SAC-solvability (because it is a sub-pattern of T4), and we show that if the induction started from V_2 breaks then $\hat{\mathrm{M}}$ must occur somewhere - a win-win situation.

▶ **Lemma 9.** *Let $I = (X, D, C) \in \mathrm{CSP}(\overline{R7\text{-}})$ be singleton arc consistent. Let $x \in X$ be such that $\hat{M}$ occurs on (y, x, z) with x the middle variable and v be the value in $D(x)$ that is the meet point of the two positive edges. Then every constraint whose scope contains a variable in $S_{(P_{xv})}$ is trivial in I_{xv}, except possibly $R(y, z)$.*

▶ **Lemma 10.** *Let $I = (X, D, C) \in \mathrm{CSP}(\overline{\hat{M}}) \cap \mathrm{CSP}(\overline{R7\text{-}})$ be singleton arc consistent. Let $x \in X$ be such that V_2 occurs on (y, x, z) with x the middle variable and v be the value in $D(x)$ that is the meet point of the two positive edges. Then every constraint whose scope contains a variable in $S_{(P_{xv})}$ is trivial in I_{xv}, except possibly $R(y, z)$.*

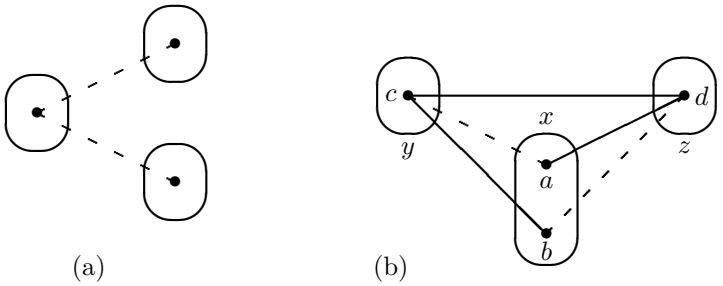


Figure 8 (a) The pattern V^- and (b) the associated broken-triangle pattern (BTP).

▶ **Theorem 11.** $\mathrm{CSP}(\overline{R7\text{-}})$ *is solved by singleton arc consistency.*

7 Tractability of Q2 and R5

For our last two proofs of SAC-decidability, we depart from the trace technique. Our fundamental goal, however, remains the same: find an operation which shrinks the instance without altering satisfiability, introducing the pattern or losing singleton arc consistency. For Q2 this operation is BTP-merging [19] and for R5 it is removing constraints.

Consider the pattern V^- shown in Figure 8(a). We say that V^- occurs at point a or at variable x if $a \in D(x)$ is the central point of the pattern in the instance. The pattern V^- is known to be tractable since all instances in $\mathrm{CSP}(\overline{V^-})$ satisfy the joint-winner property [22]. However, we show a slightly different result, namely that singleton arc consistency is sufficient to solve instances in which V^- only occurs at degree-2 variables.

▶ **Lemma 12.** *Instances in which V^- only occurs at degree-2 variables are solved by singleton arc consistency.*

Two values $a, b \in D(x)$ are *BTP-mergeable* [19] if there are not two other distinct variables $y, z \neq x$ such that $\exists c \in D(y)$, $\exists d \in D(z)$ with ad, bc, cd positive edges and ac, bd negative edges as shown in Figure 8(b). The *BTP-merging* operation consists in merging two BTP-mergeable points $a, b \in D(x)$: the points a, b are replaced by a new point c in $D(x)$ such that for all other variables $w \neq x$ and for all $d \in D(w)$, cd is a positive edge if at least one of ad, bd was a positive edge (a negative edge otherwise). BTP-merging preserves satisfiability [19].

▶ **Lemma 13.** *Let P be a pattern in which no point occurs in more than one positive edge. Then the BTP-merging operation cannot introduce the pattern P in an instance $I \in \mathrm{CSP}(\overline{P})$.*

Since Q2 has no point which occurs in more than one positive edge, we can deduce from Lemma 13 that Q2 cannot be introduced by BTP-merging. We then combine this property with Lemma 12 by proving that V^- can only occur at degree-2 variables in any instance of $\mathrm{CSP}(\overline{Q2})$ with no BTP-mergeable values.

▶ **Theorem 14.** $\mathrm{CSP}(\overline{Q2})$ *is solved by singleton arc consistency.*

That only leaves R5. Removing constraints cannot introduce R5 because it is a monotone pattern, so we can apply repeatedly the following lemma to obtain our last result.

▶ **Lemma 15.** *If the pattern R5 does not occur in a singleton arc consistent binary CSP instance I, then removing any constraint leaves the satisfiability of I invariant.*

▶ **Theorem 16.** $\mathrm{CSP}(\overline{R5})$ *is solved by singleton arc consistency.*

Note that Lemma 15 is technically true for all SAC-solvable patterns (not only R5); this is simply the only case where we are able to prove it directly.

8 Conclusion

We have established SAC-solvability of five novel classes of binary CSPs defined by a forbidden pattern, three of which are generalisations of 2SAT. For monotone patterns (defining classes of CSPs closed under removing constraints), there remains only a relatively small number of irreducible patterns whose SAC-solvability is still open. In addition to settling the remaining patterns, a possible line of future work is to study *sets* of patterns or partially-ordered patterns [23] that give rise to SAC-solvable (monotone) classes of CSPs.

References

1 Albert Atserias, Andrei A. Bulatov, and Víctor Dalmau. On the power of k -consistency. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, volume 4596 of *Lecture Notes in Computer Science*, pages 279–290. Springer, 2007. doi:10.1007/978-3-540-73420-8_26.

2 Libor Barto and Marcin Kozik. Constraint satisfaction problems solvable by local consistency methods. *J. ACM*, 61(1):3:1–3:19, 2014. doi:10.1145/2556646.

3 Nicolas Beldiceanu, Mats Carlsson, and Jean-Xavier Rampon. *Global Constraint Catalog*, 2017. http://sofdem.github.io/gccat/gccat/titlepage.html.

4 Joel Berman, Paweł Idziak, Petar Marković, Ralph McKenzie, Matthew Valeriote, and Ross Willard. Varieties with few subalgebras of powers. *Transactions of the American Mathematical Society*, 362(3):1445–1473, 2010. doi:10.1090/S0002-9947-09-04874-0.

5 Christian Bessiere and Romuald Debruyne. Theoretical analysis of singleton arc consistency and its extensions. *Artif. Intell.*, 172(1):29–41, 2008. doi:10.1016/j.artint.2007.09.001.

6 Christian Bessière, Jean-Charles Régin, Roland H. C. Yap, and Yuanlin Zhang. An optimal coarse-grained arc consistency algorithm. *Artif. Intell.*, 165(2):165–185, 2005. doi:10.1016/j.artint.2005.02.004.

7 Sally C. Brailsford, Chris N. Potts, and Barbara M. Smith. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, 119(3):557–581, 1999. doi:10.1016/S0377-2217(98)00364-6.

8 Andrei Bulatov. Bounded relational width, 2009. Unpublished manuscript.

9 Andrei A. Bulatov. A dichotomy theorem for nonuniform csps. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 319–330. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.37.

10 Andrei A. Bulatov and Víctor Dalmau. A simple algorithm for mal'tsev constraints. *SIAM J. Comput.*, 36(1):16–27, 2006. doi:10.1137/050628957.

11 Andrei A. Bulatov, Peter Jeavons, and Andrei A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, 2005. doi:10.1137/S0097539700376676.

12 Clément Carbonnel, David A. Cohen, Martin C. Cooper, and Stanislav Zivny. On singleton arc consistency for CSPs defined by monotone patterns, April 2017. URL: http://arxiv.org/abs/1704.06215.

13 Hubie Chen, Víctor Dalmau, and Berit Grußien. Arc consistency and friends. *J. Log. Comput.*, 23(1):87–108, 2013. doi:10.1093/logcom/exr039.

14 Cheng-Chung Cheng and Stephen F. Smith. Applying constraint satisfaction techniques to job shop scheduling. *Annals of Operations Research*, 70(0):327–357, 1997. doi:10.1023/A:1018934507395.

15 David A. Cohen, Martin C. Cooper, Páidí Creed, Dániel Marx, and András Z. Salamon. The tractability of CSP classes defined by forbidden patterns. *J. Artif. Intell. Res.*, 45:47–78, 2012. doi:10.1613/jair.3651.

16 David A. Cohen, Martin C. Cooper, Guillaume Escamocher, and Stanislav Zivny. Variable and value elimination in binary constraint satisfaction via forbidden patterns. *J. Comput. Syst. Sci.*, 81(7):1127–1143, 2015. doi:10.1016/j.jcss.2015.02.001.

17 David A. Cohen and Peter G. Jeavons. The power of propagation: when GAC is enough. *Constraints*, 22(1):3–23, 2017. doi:10.1007/s10601-016-9251-0.

18 Martin C. Cooper, David A. Cohen, and Peter Jeavons. Characterising tractable constraints. *Artif. Intell.*, 65(2):347–361, 1994. doi:10.1016/0004-3702(94)90021-3.

19 Martin C. Cooper, Aymeric Duchein, Achref El Mouelhi, Guillaume Escamocher, Cyril Terrioux, and Bruno Zanuttini. Broken triangles: From value merging to a tractable class of general-arity constraint satisfaction problems. *Artif. Intell.*, 234:196–218, 2016. doi:10.1016/j.artint.2016.02.001.

20 Martin C. Cooper and Guillaume Escamocher. Characterising the complexity of constraint satisfaction problems defined by 2-constraint forbidden patterns. *Discrete Applied Mathematics*, 184:89–113, 2015. doi:10.1016/j.dam.2014.10.035.

21 Martin C. Cooper, Peter G. Jeavons, and András Z. Salamon. Generalizing constraint satisfaction on trees: Hybrid tractability and variable elimination. *Artif. Intell.*, 174(9-10):570–584, 2010. doi:10.1016/j.artint.2010.03.002.

22 Martin C. Cooper and Stanislav Zivny. Hybrid tractability of valued constraint problems. *Artif. Intell.*, 175(9-10):1555–1569, 2011. doi:10.1016/j.artint.2011.02.003.

23 Martin C. Cooper and Stanislav Zivny. The power of arc consistency for csps defined by partially-ordered forbidden patterns. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 652–661. ACM, 2016. doi:10.1145/2933575.2933587.

24 Eugene C. Freuder. A sufficient condition for backtrack-free search. *J. ACM*, 29(1):24–32, 1982. doi:10.1145/322290.322292.

25 Eugene C. Freuder. Eliminating Interchangeable Values in Constraint Satisfaction Problems. In *Proceedings of AAAI-91*, pages 227–233, 1991. URL: http://www.aaai.org/Library/AAAI/1991/aaai91-036.php.

26 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1:1–1:24, 2007. doi:10.1145/1206035.1206036.

27 Pawel M. Idziak, Petar Markovic, Ralph McKenzie, Matthew Valeriote, and Ross Willard. Tractability and learnability arising from algebras with few subpowers. *SIAM J. Comput.*, 39(7):3023–3037, 2010. doi:10.1137/090775646.

28 Marcin Kozik. Weak consistency notions for all the csps of bounded width. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 633–641. ACM, 2016. doi:10.1145/2933575.2934510.

29 Norman Lim, Shikharesh Majumdar, and Peter Ashwood-Smith. A constraint programming-based resource management technique for processing mapreduce jobs with

slas on clouds. In *43rd International Conference on Parallel Processing, ICPP 2014, Minneapolis, MN, USA, September 9-12, 2014*, pages 411–421. IEEE Computer Society, 2014. doi:10.1109/ICPP.2014.50.

30 Roger Mohr and Thomas C. Henderson. Arc and path consistency revisited. *Artif. Intell.*, 28(2):225–233, 1986. doi:10.1016/0004-3702(86)90083-4.

31 Cemalettin Ozturk and M. Arslan Ornek. Optimisation and constraint based heuristic methods for advanced planning and scheduling systems. *International Journal of Industrial Engineering: Theory, Applications and Practice*, 23(1):26–48, 2016. URL: http://journals.sfu.ca/ijietap/index.php/ijie/article/view/1930.

32 Pinar Senkul and Ismail Hakki Toroslu. An architecture for workflow scheduling under resource allocation constraints. *Inf. Syst.*, 30(5):399–422, 2005. doi:10.1016/j.is.2004.03.003.

33 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 331–342. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.38.

The Firing Squad Problem Revisited

Bernadette Charron-Bost
École polytechnique, CNRS, 91128 Palaiseau, France
charron@lix.polytechnique.fr

Shlomo Moran
Department of Computer Science, Technion, Haifa, Israel 32000
moran@cs.technion.ac.il

Abstract

In the classical firing squad problem, an unknown number of nodes represented by identical finite state machines is arranged on a line and in each time unit each node may change its state according to its neighbors' states. Initially all nodes are passive, except one specific node located at an end of the line, which issues a fire command. This command needs to be propagated to all other nodes, so that eventually all nodes simultaneously enter some designated "firing" state.

A natural extension of the firing squad problem, introduced in this paper, allows each node to postpone its participation in the squad for an arbitrary time, possibly forever, and firing is allowed only after all nodes decided to participate. This variant is highly relevant in the context of decentralized distributed computing, where processes have to coordinate for initiating various tasks simultaneously.

The main goal of this paper is to study the above variant of the firing squad problem under the assumptions that the nodes are *infinite* state machines, and that the inter-node communication links can be changed arbitrarily in each time unit, i.e., are defined by a *dynamic graph*. In this setting, we study the following fundamental question: what connectivity requirements enable a solution to the firing squad problem?

Our main result is an exact characterization of the dynamic graphs for which the firing squad problem can be solved. When restricted to static directed graphs, this characterization implies that the problem can be solved if and only if the graph is strongly connected. We also discuss how information on the number of nodes or on the diameter of the network, and the use of randomization, can improve the solutions to the problem.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Theory of computation → Dynamic graph algorithms, Theory of computation → Distributed algorithms

Keywords and phrases Synchronization, Detection, Simultaneity, Dynamic Networks

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.20

1 Introduction

Many distributed algorithms assume a *synchronous networked system*, in which computation is divided into *synchronized rounds* that are communication closed layers: any message sent at some round can be received only at that round. In this model it is typically assumed that each execution of an algorithm is started by all nodes simultaneously, i.e., at the same round. For instance, most of synchronous consensus algorithms (eg., [21, 12, 23]), as well as many distributed algorithms for dynamic networks (eg., [16, 17]) require synchronous starts.

In this paper, we justify this assumption of synchronous starts for dynamic networks with no central control that monitors the node activities, but with sufficient connectivity assumptions. Specifically, we study a generalization of the associated synchronization problem, classically referred to as the *firing squad problem*. This generalization considers

a communication network of unknown size, in which messages are delivered along a set of edges which may change in each round. All nodes are initially *passive*, and a node becomes *active* upon receiving a *start signal* at an unpredictable time. We stress that receiving a message from an active node is not necessarily considered as a start signal. The goal is then to guarantee that the nodes synchronize by *firing* - i.e., entering a designated state for the first time - simultaneously if and only if *all* nodes are eventually active. Formally, the following must be satisfied:

FS1 (Validity): A node fires if and only if all nodes have received start signals.

FS2 (Simultaneity): All the nodes that fire, fire at the same round.

As a basic synchronization abstraction, the fulfillment of FS1 and FS2 above can be used in various types of situations to guarantee simultaneity: for distributed initiation (to force nodes to begin some computation in unison), in real-time processing (where nodes have to carry out some external actions simultaneously), or for distributed termination (to guarantee that nodes complete their computation at the same round). Another typical scenario that requires FS1 and FS2 is when some algorithm needs to be executed several times in a row, and the $i+1$-st execution should be started simultaneously, after all nodes terminated the i-th execution (see e.g., [5]).

It is easy to see that when the communication graph is permanently complete, the firing squad problem can be solved in one round after all nodes are active. At the opposite scenario, the problem is clearly unsolvable if some node is permanently isolated. This demonstrates a strong correlation between the solvability and complexity of the firing squad problem, and the connectivity of the network. The primary aim of this paper is to explore this relation.

The firing squad problem was originally studied in the context of automata theory (eg., [18, 19]). This model considers a finite but unknown number n of nodes which are connected in a line (or in some other specific topologies in more recent works – see eg., [8]). Nodes are identical *finite* state machines (whose number of states is independent of n), and at each time unit each node changes its state according to the states of its neighbors on the line. A start signal is given to a node located at one end of the line - the “general" - and then is propagated to the rest of the nodes so that all nodes have eventually to fire simultaneously. It should be noted that the above model assumes *diffusive start signals*, for which the timing of start signals is not arbitrary: upon the receipt of a message from an active node, a passive node becomes active, i.e., receiving such a message is considered as a start signal. The main challenges in this model are to reduce the number of states of the finite state machine and the time required to reach the firing state.

A natural question raised at this point is then the following: considering that nodes are no longer restricted to be finite state machines, but possess a full computational power (equivalent to that of a Turing machine), what are the connectivity properties that are needed to solve the firing squad problem?

It should be noted that the firing squad problem has also been studied in the context of fault tolerant distributed computations (eg., [3, 12, 7]), and more recently in the context of self-stabilization (eg., see [10]). This model also assumes that each node has a full computational power, but otherwise the setting of the problem is different: Nodes are connected by a complete graph, and thus the number of nodes n is given. At most f nodes may be faulty, for various types of Byzantine faults. This implies a permanent complete connectivity between the non-faulty nodes, and arbitrary connectivity of all other links. Besides, due to the unpredictable behavior of faulty nodes, the simultaneity condition and to a larger extent the validity condition in this model ought to be drastically weakened: eg., it is only required that all non-faulty nodes eventually fire simultaneously. Finally, the study of the problem is strictly limited to diffusive start signals.

Contribution. In this paper we consider a set of an unknown number n of nodes possessing full computational power. Nodes have distinct identities which are not mutually known, but otherwise they run identical codes (in some precise sense that is discussed later). The inter-node communication is modeled by a *dynamic graph*, i.e., at each round, nodes communicate along directed edges of an arbitrary communication graph which may change continually and unpredictably from one round to the next. Communication is done by having each node broadcast at each round a message along the unknown set of its outgoing edges in this round. We examine various connectivity properties that hold, not necessary round by round, but globally over finite periods of consecutive rounds. In particular, these properties do not imply any stability of the links, as opposed to the failure model of at most f faulty nodes that guarantees a stable clique of size $n - f$, or several models of dynamic networks in distributed computing (eg., see [16, 1, 22]) that assume the existence of a stable spanning tree in the network over every T consecutive rounds.

The main contribution of this paper is a characterization of the connectivity properties that enable to solve the firing squad problem in dynamic (and hence also in static) graphs. On the positive side, we show that if the dynamic graph is guaranteed to be connected within each period of T consecutive rounds, where the constant T is given, then the problem is solvable in time which is at most linear in the (unknown) network size. On the negative side, we show that under the sole assumption that such a constant T exists but is unknown, the problem becomes unsolvable. Moreover, the problem remains unsolvable in this case even when the number of nodes in the network is given and even in the restricted model of diffusive start signals. The above results imply that the firing squad problem is solved for a *static* directed graph if and only if it is strongly connected.

Our solution is obtained by combining two basic procedures: the first implements local virtual clocks whose values cannot exceed the *diameter of the dynamic graph* unless all nodes are active, and the second collects the identities of all nodes in the network. The idea is then that a node fires when the value of its virtual clock is sufficiently large compared to the number of active nodes it has heard of so far.

We also show that if an upper bound D on the diameter of the dynamic graph is given, then the problem is solvable in time linear in D. This solution is applicable to anonymous networks, where nodes have no identities, and it uses much shorter messages. We conclude by showing that when a polynomial bound on the network size is given, the use of randomization can substantially reduce messages size while preserving a linear time complexity.

For space consideration, some proofs are omitted in this version.

2 The Model

2.1 Distributed computations in the dynamic graphs model

We consider a networked system with a *fixed* set of n nodes. Nodes have unique identifiers, and the set of identifiers is denoted by V. The identities of the nodes are not mutually known, and the network size n is unknown as well. Nodes may also ignore their own identities, in which case the network is said to be *anonymous*. Furthermore, nodes run identical programs, i.e., programs do not depend on node identities (see the discussion in Section 5).

Computation proceeds in *synchronized rounds*, which are communication closed in the sense that no node receives messages in round t that are sent in a round different from t. In round t ($t = 1, 2 \dots$), each node attempts to send messages to all nodes, receives messages from some nodes, and finally goes to its next state and proceeds to round $t + 1$. The round number t is used for a reference, but is unknown to the nodes.

In every *run* of an algorithm, each node u is initially *passive*: it is part of the network, but sends only heartbeats – that we call *null* messages – and does not change its state. Then it either becomes *active* by receiving a unique *start signal* at the beginning of some round $s_u \geqslant 1$, or remains passive forever – in which case we let $s_u = +\infty$. A run is *active* if all nodes are eventually active.

Upon the receipt of its start signal, node u sets up its local variables (with its initial state) and starts executing its program. For any of its local variables x_u, the value of x_u at the beginning of round t is denoted by $x_u(t)$. Thus $x_u(t)$ is undefined for $t < s_u$.

Communications that occur at round t are modeled by a directed graph $\mathbb{G}(t) = (V, E_t)$ that may change from round to round. We assume a self-loop at each node in all the graphs $\mathbb{G}(t)$ since any node can communicate with itself instantaneously.

The sequence of directed graphs $\mathbb{G} = (\mathbb{G}(t))_{t\in\mathbb{N}}$ is called a *dynamic graph* [4]. It can be decided ahead of time, by an online adversary, or endogenously as in *influence systems* [6]. Similarly, the way start signals are generated is left totally arbitrary: a node may receive an *external* start signal coming from outside, or it may receive a start signal relayed by some active node. In particular, there may be more than one external start signal in the network, and start signals may be not correlated to the dynamic graph.

A run of a firing squad algorithm is entirely determined by the dynamic graph $\mathbb{G} = (\mathbb{G}(t))_{t\in\mathbb{N}}$ and by the list $\mathbb{S} = (s_u)_{u\in V}$ of rounds at which nodes become active. We denote by $\mathbb{G}^*(t) = (V, E_t^*)$ the directed graph of edges in E_t connecting two nodes which are active in round t. The sets of u's incoming neighbors (in-neighbors for short) in the directed graphs $\mathbb{G}(t)$ and $\mathbb{G}^*(t)$ are denoted by $\mathrm{In}_u(t)$ and $\mathrm{In}_u^*(t)$, respectively.

Let $\mathcal{D}$ be a set of dynamic graphs. We say that an algorithm A *solves the firing squad problem for* $\mathcal{D}$ if for each $\mathbb{G} \in \mathcal{D}$ and each scheduling of start signals $\mathbb{S}$, the run of A defined by $\mathbb{G}$ and $\mathbb{S}$ satisfies FS1 and FS2. The firing squad problem is *solvable for* $\mathcal{D}$ if there is an algorithm that solves it for $\mathcal{D}$.

2.2 Paths and broken paths in a dynamic graph

Let us first recall that the *product* of two directed graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, denoted $G_1 \circ G_2$, is the directed graph with the set of nodes V and with an edge (u, v) if there exists $w \in V$ such that $(u, w) \in E_1$ and $(w, v) \in E_2$.

For any dynamic graph $\mathbb{G}$ and any integers $t' > t \geqslant 1$, we let $\mathbb{G}(t : t') = \mathbb{G}(t) \circ \cdots \circ \mathbb{G}(t')$. By convention, $\mathbb{G}(t : t) = \mathbb{G}(t)$, and $\mathbb{G}(t : t')$ is the directed graph with only a self-loop at each node when $t' < t$. We also use the notation $\mathbb{G}(I)$ instead of $\mathbb{G}(t : t')$ when I is the integer interval $[t, t']$.

We now fix a run of a firing squad algorithm, with the dynamic graph $\mathbb{G}$ and the scheduling of start signals $\mathbb{S}$ which, as above, determine the dynamic graph $\mathbb{G}^*$. The sets of u's in-neighbors in $\mathbb{G}(t : t')$ and in $\mathbb{G}^*(t : t')$ are denoted by $\mathrm{In}_u(t : t')$ and $\mathrm{In}_u^*(t : t')$, respectively, or by $\mathrm{In}_u(I)$ and $\mathrm{In}_u^*(I)$ for short when $I = [t, t']$.

Let t and t' be two positive integers such that $t' \geqslant t$; a $v{\sim}u$ *path in the interval* $[t, t']$ is any sequence $P = (v_0 = v, v_1, \ldots, v_m = u)$ with $m = t' - t + 1$ and (v_k, v_{k+1}) is an edge of $\mathbb{G}(t + k)$ for each $k = 0, \ldots, m - 1$. Hence there exists a $v{\sim}u$ path in the interval $[t, t']$ if and only if $v \in \mathrm{In}_u(t : t')$. The path P is said to be *broken* if one of its edges (v_k, v_{k+1}) is not in $\mathbb{G}^*(t + k)$.

2.3 Delayed connectivity of a dynamic graph

Let us recall that a directed graph is *strongly connected* if for each pair of nodes u, v there is a directed path from u to v. For $c \geqslant 1$, *c strong connectivity* is then defined by (see, e.g., [9]):

▶ **Definition 1.** Let $G = (V, E)$ be a directed graph and let $c < |V|$ be a positive integer. We say that G is *c strongly connected* if G remains strongly connected whenever less than c nodes are removed from G.

Note that a directed graph is strongly connected if and only if it is 1 strongly connected.

▶ **Definition 2.** A dynamic graph $\mathbb{G}$ is *continuously c strongly connected* if each directed graph $\mathbb{G}(t)$ is c strongly connected.

Next we extend the above definition to bounded-length intervals of dynamic graphs.

▶ **Definition 3.** Let c, T be two positive integers. The dynamic graph $\mathbb{G}$ is *c connected with delay T* if for every positive integer t, the directed graph $\mathbb{G}(t : t + T - 1)$ is c strongly connected. When $c = 1$, we use the abbreviation *connected with delay T*.

Finally, we present our weakest connectivity assumption for dynamic graphs.

▶ **Definition 4.** A dynamic graph $\mathbb{G}$ is said to be *eventually connected* if for any positive integer t, there exists $t' \geqslant t$ such that $\mathbb{G}(t : t')$ is strongly connected.

Using the connectivity properties of dynamic graphs defined above, we then characterize the connectivity properties that enable solutions to the firing squad problem. For a positive integer T, $\mathcal{D}_T$ denotes the set of dynamic graphs which are connected with delay T. The union $\mathcal{D}_B = \bigcup_{T=1}^{\infty} \mathcal{D}_T$ is the set of dynamic graphs with *bounded delay connectivity* and $\mathcal{D}_E$ denotes the set of eventually connected dynamic graphs. The relations among the above sets of dynamic graphs are thus given by the strict inclusions

$$\mathcal{D}_1 \subset \mathcal{D}_2 \subset \cdots \subset \mathcal{D}_T \subset \mathcal{D}_{T+1} \subset \cdots \subset \mathcal{D}_B \subset \mathcal{D}_E.$$

In the next sections, we show that the firing squad problem is not solvable for $\mathcal{D}_B$ (and hence also for $\mathcal{D}_E$), but for each positive integer T, it is solvable for $\mathcal{D}_T$.

3 Bounded Delay Connectivity is not Enough

In this section we show that the firing squad problem is not solvable for the set $\mathcal{D}_B$ of the dynamic graphs with bounded delay connectivity, even if the network size, n, is given. Specifically, we show that for this set of dynamic graphs, the validity condition FS1 can be achieved if and only if n is given, and the firing squad problem (i.e., FS1 plus FS2) cannot be solved even if n is given.

Interestingly, these two impossibility results still hold for the original model of diffusive start signals and when all communication graphs are bidirectional.

▶ **Proposition 5.** *For the set of dynamic graphs with bounded delay connectivity $\mathcal{D}_B$, the validity condition FS1 can be achieved if the network size n is given, but cannot be achieved if it is given that the network size is either n or $n + 1$.*

Next we show that there is no algorithm that solves the firing squad problem for $\mathcal{D}_B$, even if the number of nodes in the dynamic graph is given. This demonstrates that adding the simultaneity condition FS2 to the validity condition FS1 makes the problem strictly harder and that the knowledge of the size of the network does not help in the sole context of bounded delay connectivity.

▶ **Theorem 6.** *The firing squad problem is not solvable for the set $\mathcal{D}_B$ of dynamic graphs with bounded delay connectivity, even if the size of the network n is given.*

Proof. By contradiction, suppose that there is an algorithm A solving the firing squad problem in any dynamic graph with n nodes and with bounded delay connectivity, and let V be a set of $n > 1$ nodes.

Let u, v be two distinct nodes in V, and for $x \in \{v, u\}$ let G_x be the graph consisting of a complete graph over $V \setminus \{x\}$ plus the self loop (x, x). Let further $I = (V, E_I)$ denote the directed graph with only a self-loop at each node, i.e., $E_I = \{(v, v) : v \in V\}$.

We consider the run of A in which all nodes are active in the first round, and with the dynamic graph consisting of alternating sequence of directed graphs $\mathbb{G} = (G_u, G_v, G_u, G_v, \dots)$. Clearly, $\mathbb{G} \in \mathcal{D}_B$, and thus by FS1-2, all nodes fire at the same round t_F.

Now assume that $\mathbb{G}(t_F) = G_u$ (the case $\mathbb{G}(t_F) = G_v$ is similar). From the viewpoint of u, $\mathbb{G}$ is indistinguishable up to round t_F from the dynamic graph $\mathbb{G}^1$ that is similar to $\mathbb{G}$ except at round t_F where $\mathbb{G}^1(t_F) = I$. Hence u also fires at round t_F with the dynamic graph $\mathbb{G}^1$. Since $\mathbb{G}^1 \in \mathcal{D}_B$, all other nodes also fire at round t_F with $\mathbb{G}^1$. Using a similar argument, we get that from the viewpoint of v, $\mathbb{G}^1$ is indistinguishable up to round t_F from the dynamic graph $\mathbb{G}^2$ that is similar to $\mathbb{G}^1$ except at round $t_F - 1$, in which $\mathbb{G}^2(t_F - 1) = I$. Hence with $\mathbb{G}^2$, all nodes fire at round t_F as well.

By repeating this argument t_F times, we show that all nodes fire at round t_F in the run of A with start signals all received in the first round, and the dynamic graph $\mathbb{G}^{t_F} = (I, \dots, I, G_v, G_u, G_v, G_u, \dots)$. From the viewpoint of any node $v \neq u$, the latter run is indistinguishable up to round t_F from the run with the same dynamic graph $\mathbb{G}^{t_F}$ and where all nodes are active from round one except node u which is passive forever. All nodes other than u fire at round t_F, violating FS1 - a contradiction. ◀

4 Firing with a Bounded Diameter

As a first step towards our main positive result, which solves the firing squad problem in dynamic graphs that are c connected with delay T, we present a solution in the case that a finite bound on the *diameter* of the dynamic graph is given. We start with some definitions.

Let $\mathbb{G} = (\mathbb{G}(t))_{t \in \mathbb{N}}$ be a dynamic graph. The *distance from node v to node w at time t*, denoted $d_t(v, w)$, is defined as the minimum positive integer δ such that there is a $v \sim w$ path in the interval $[t, t + \delta - 1]$. If for any $t' \geqslant t$ there is no $v \sim w$ path in the interval $[t, t']$, then conventionally $d_t(v, w) = +\infty$.

The *diameter* of the dynamic graph $\mathbb{G}$ is then defined as the minimum positive integer d such that for any positive integer t, the directed graph $\mathbb{G}(t : t + d - 1)$ is complete, or infinity if there is no such integer, namely $\operatorname{diam}(\mathbb{G}) = \sup_{t \geqslant 1,\, v,w \in V^2} d_t(v, w)$.

Let $\mathcal{D}$ be a set of dynamic graphs, and assume that a finite bound D on the diameters of the dynamic graphs in $\mathcal{D}$ is given. Then a solution to the firing squad problem is enabled by using local virtual clocks whose values may reach D only if all nodes are active. Moreover, if some virtual clock is set to D, then all virtual clocks are set to D at the same round. The corresponding algorithm, denoted A_D, does not use identifiers, and the computation and storage capabilities of the nodes do not grow with the network size. More precisely, its time complexity is in $O(D)$ and it uses only $O(\log(D))$ bits per message.

Notation. In the pseudo-codes of all our algorithms, M_u^* denotes the multiset of non-null messages received by u in the current round. Thus M_u^* at round t is the multiset of messages sent to u by the nodes in $\mathrm{In}_u^*(t)$. If non-null messages are vectors of some size, then $M_u^{*(i)}$ denotes the multiset of the i-th entries of the messages in M_u^*.

Algorithm 1: Algorithm A_D, firing with diameter at most D.

Initialization:
1: $r_u \in \mathbb{N}$, initially 0
In each round t **do:**
2: send $\langle r_u \rangle$ to all processes and receive one message from each in-neighbor
3: **if** at least one received message is null **then**
4: $\quad r_u \leftarrow 0$
5: **else**
6: $\quad r_u \leftarrow 1 + \min_{r \in M_u^*}(r)$
7: **end if**
8: **if** $r_u \geqslant D$ **then**
9: $\quad$ Fire
10: **end if**

We begin the correctness proof of the algorithm A_D by two useful lemmas about the way the virtual clocks r_u's evolve, whatever the connectivity properties of dynamic graphs are.

▶ **Lemma 7.** *Assume that $t < t'$ and $s_u \leqslant t'$. Then $r_u(t')$ is defined and:*
1. *If there exists a broken path ending at u in the interval $[t, t'-1]$, then $r_u(t') \leqslant t' - t - 1$.*
2. *Otherwise, for every $v \in In_u(t : t'-1)$ it holds that $r_v(t)$ is defined and $r_u(t') \leqslant r_v(t) + t' - t$.*

▶ **Lemma 8.** *For every node u and at every round $t \geqslant s_{\max} = \max_{v \in V}(s_v)$ of an active run, we have $r_u(t) \geqslant t - s_{\max}$. Moreover, if $t \geqslant s_{\max} + 1$ and $\mathrm{In}_u(s_{\max} : t-1)$ contains a node v such that $s_v = s_{\max}$, then $r_u(t) = t - s_{\max}$.*

From the two above lemmas, we can prove the correctness of the algorithm A_D:

▶ **Theorem 9.** *The algorithm A_D solves the firing squad problem for any set of dynamic graphs with diameters at most D. Moreover, all nodes in an active run of the algorithm fire exactly D rounds after all nodes have become active and use messages of size $O(\log D)$.*

Observe that the diameter of any connected dynamic graph with n nodes is at most $n-1$. Thus one immediate spinoff of Theorem 9 is the following corollary, which when an upper bound N on the network size is given, provides a solution to the firing squad problem that uses messages of size $O(\log(N))$.

▶ **Corollary 10.** *If nodes have an upper bound N of the network size, the firing squad problem can be solved in any continuously strongly connected dynamic graph in N rounds after all nodes have become active using only $O(\log(N))$ bits per message.*

5 Firing with T Delayed Connectivity

We now present the algorithm $B_{c,T}$ that show that it solves the firing squad problem in linear time for dynamic graphs that are c connected with delay T while no bound on the diameter or the size of the network is given.

The algorithm $B_{c,T}$ uses the same virtual clocks r_u as the previous algorithm A_D. Moreover, each node u collects the identities of the active nodes which u had heard of in a variable HO_u. Then node u fires when its virtual clock r_u is large enough compared to the size of its HO_u set.

Algorithm 2: Algorithm $B_{c,T}$, firing with T delayed connectivity.

Initialization:
1: $r_u \in \mathbb{N}$, initially 0
2: $HO_u \subseteq V$, initially $\{u\}$
In each round t **do:**
3: send $\langle r_u, HO_u \rangle$ to all processes and receive one message from each in-neighbor
4: **if** at least one received message is null **then**
5: $\quad r_u \leftarrow 0$
6: **else**
7: $\quad r_u \leftarrow 1 + \min_{r \in M_u^{*(1)}}(r)$
8: **end if**
9: $HO_u \leftarrow \cup_{HO \in M_u^{*(2)}} HO$
10: **if** $|HO_u| \leqslant \left\lceil \frac{c}{T}(r_u + 2) \right\rceil - 2c$ **then**
11: $\quad$ Fire
12: **end if**

A similar idea was first used in [14], and also later in *early stopping consensus algorithms* [11, 13] and in the counting algorithm of [16], but with different virtual clocks. This technique requires distinct node identifiers and long messages since each node u broadcasts HO_u in each round.

The following lemma is needed for the analysis of the algorithm $B_{c,T}$.

▶ **Lemma 11.** *If $G = (V, E)$ is c strongly connected, then for any non-empty subset $S \subseteq V$, the following holds:*

$$|\Gamma_{\text{in}}(S) \setminus S| \geqslant \min(c, |\overline{S}|) \tag{1}$$

where $\Gamma_{\text{in}}(S)$ denotes the set of in-neighbors of S in G, and $\overline{S} = V \setminus S$.

It can be shown that the converse of Lemma 11 also holds. Moreover, the set $\Gamma_{\text{out}}(S)$ of out-neighbors of S can be substituted for $\Gamma_{\text{in}}(S)$ in Lemma 11 since any directed graph G is c strongly connected if and only if its transpose G^T is. Using this out-variant of Lemma 11 and an easy induction, we check that the diameter of a dynamic graph that is c connected with delay T is bounded by $T\left\lfloor 1 + \frac{n-2}{c} \right\rfloor$.

The correctness proof of the algorithm $B_{c,T}$ then relies on the following key technical lemma.

▶ **Lemma 12.** *In each run of the algorithm $B_{c,T}$ on a dynamic graph $\mathbb{G}$ which is c connected with delay T, for each node u and each round $t \geqslant s_u$, it holds that $r_u(t)$ and $HO_u(t)$ are defined and*

$$|HO_u(t)| \geqslant \min\left((1 - 2c) + \frac{c}{T}(r_u(t) + 2), n \right) \quad . \tag{2}$$

Proof. If $t = 1$, then $s_u = 1$, $HO_u(t) = \{u\}$, $r_u(t) = 0$, and the lemma holds.

So assume now that $t \geqslant 2$, and let $a, b \in \mathbb{N}$ satisfy $t = aT + b$ with $1 \leqslant b \leqslant T$. We split the interval $[1, t-1]$ into $a+1$ sub-intervals $I_a, I_{a-1}, \ldots, I_1, I_0$ as follows:
1. if $b = 1$, then I_0 is the empty interval, else $I_0 = [t - b + 1, t - 1]$;
2. for $0 < i \leqslant a$, we set $I_i = [t - b - iT + 1, t - b - (i-1)T]$.
We check that $|I_0| = b - 1 < T$, and $|I_i| = T$ for $i > 0$. All the intervals I_i are thus non-empty, except I_0 that is empty if and only if $b = 1$.

Then by induction, we construct a sequence of at most $a + 2$ sets of nodes $S_{-1}, S_0, \ldots, S_k$ as follows:

1. $S_{-1} = \{u\}$.
2. Suppose that $S_{-1}, \ldots, S_i$, $-1 \leqslant i \leqslant a$, are constructed.
   - **a.** If $i = a$, then the construction stops.
   - **b.** Otherwise, $-1 \leqslant i \leqslant a-1$. We let $H_{i+1} = \mathbb{G}\big(I_{i+1}\big)$ and we distinguish three cases.
     - **i.** $i \geqslant 0$ and H_{i+1} contains no edge (w, v) such that $w \notin S_i$ and $v \in S_i$. Then the construction stops.
     - **ii.** H_{i+1} contains an edge (w, v) such that $w \notin S_i$ and $v \in S_i$, and there exists a $w{\sim}v$ broken path in I_{i+1}. Then the construction stops.
     - **iii.** Otherwise, we let $S_{i+1} = In_u(t-b-(i+1)T+1 : t-1) = In_u(I_{i+1} \cup \cdots \cup I_0)$, which is the union of S_i and of the set of S_i's in-neighbors in the directed graph H_{i+1}. In particular, if u has no proper in-neighbor in $H_0 = \mathbb{G}(I_0)$ (eg., if $b = 1$), then $S_0 = \{u\}$.

Let us observe that $S_{-1} \subseteq S_0$, and the sequence $\big(S_i\big)_{0 \leqslant i \leqslant k}$ is increasing. More precisely, using the T delayed c connectivity of $\mathbb{G}$ and Lemma 11, we obtain that for every index i, $1 \leqslant i \leqslant k$, if $S_i \neq V$, then $|S_i| - |S_{i-1}| \geqslant c$. By an easy induction, then we obtain the following lower bound on $|S_k|$.

▶ **Claim 13.** *If $S_k \neq V$, then the cardinality of S_k is at least $c\,k + 1$.*

Because of the way HO_u is updated (line 9 of the algorithm), we check the following claim by induction.

▶ **Claim 14.** *$HO_u(t)$ contains every set S_i for $-1 \leqslant i \leqslant k$, and in particular $S_k \subseteq HO_u(t)$.*

We now distinguish the following three exhaustive cases:

Construction terminated by (a): Since clearly $r_u(t) \leqslant t - 1 = aT + b - 1$, we have

$$(1-2c) + \frac{c}{T}\,(r_u(t)+2) \leqslant (ac+1) + \frac{c}{T}\,(b+1-2T) \leqslant ac+1 \ .$$

Moreover by Claims 13 and 14, it holds that $|HO_u(t)| \geqslant |S_k| \geqslant ac+1$. Hence

$$|HO_u(t)| \geqslant (1-2c) + \frac{c}{T}\,(r_u(t)+2) \ ,$$

which shows the lemma in this case.

Construction terminated by (b.i): In this case, $0 \leqslant k \leqslant a-1$ and so the interval I_{k+1} is defined and is of length T. Since $\mathbb{G}$ is connected with delay T, this implies that $S_k = V$. It follows that $HO_u(t) = V$ and the lemma trivially follows.

Construction terminated by (b.ii): We first observe that $S_k \neq V$. Thus by Claims 13 and 14, $|HO_u(t)| \geqslant |S_k| \geqslant ck+1$. Also, observe that the assumed $w{\sim}v$ broken path in I_{k+1} can be extended to a $w{\sim}u$ broken path in the interval $I_{k+1} \cup \cdots \cup I_0 = [t-b-(k+1)T+1, t-1]$. Since $b \leqslant T$, this implies by Lemma 7.1 that $r_u(t) \leqslant (k+2)T - 2$ or equivalently that $k \geqslant \frac{r_u(t)+2}{T} - 2$. Thus we get

$$|HO_u(t)| \geqslant ck+1 \geqslant c\left(\frac{r_u(t)+2}{T} - 2\right) + 1 = (1-2c) + \frac{c}{T}\,(r_u(t)+2) \ ,$$

which proves the lemma in this case. ◀

▶ **Theorem 15.** *The algorithm $B_{c,T}$ solves the firing squad problem for every set of dynamic graphs that are c connected with delay T. Moreover, in any active run all nodes fire in less than $\left\lceil \frac{T}{c}\,(n-1) \right\rceil + T$ rounds after all nodes have become active and they use messages of size $O(n \log n)$.*

Proof. Let us first consider a run of the algorithm in which there is a node v that is never active. Then no node ever receives a non-null message from v, and so for any node u that is active at round t, we have $|HO_u(t)| \leqslant n-1$. This implies by Lemma 12 that $|HO_u(t)| > \left\lceil \frac{c}{T}\left(r_u(t)+2\right)\right\rceil - 2c$, and hence u does not fire at round t. We conclude that no node ever fires in this run.

Let us now consider an active run of the algorithm. First, observe that by the first claim in Lemma 8 and the fact that the cardinality of each set HO_u is at most n, the condition in line 10 eventually holds at each node u.

Moreover, because of the initialization and update rules for the HO variables (lines 2 and 9), a node $v \neq u$ is in $HO_u(t+1)$ if and only if there exists a $v{\sim}u$ non-broken path in some non-empty interval $[s,t]$. Since $u \in \mathrm{In}^*_u(s_u)$, this shows that

$$HO_u(t+1) \subseteq \bigcup_{s \geqslant s_u} \mathrm{In}^*_u(s:t) \ . \tag{3}$$

Let t_0 be the first round at which the condition in line 10 holds at some node, and let u denote one such node, i.e.,

$$|HO_u(t_0+1)| \leqslant \left\lceil \frac{c}{T}\left(r_u(t_0+1)+2\right)\right\rceil - 2c \ . \tag{4}$$

From Lemma 12, we deduce that $HO_u(t_0+1) = V$. In particular, $HO_u(t_0+1)$ contains the latest activated nodes. Let v denote one such node, i.e., $s_v = s_{\max}$. By (3), there is a $v{\sim}u$ non-broken path in some interval $[s,t_0]$ with $s \geqslant s_u$. It follows that $s \geqslant s_v$. Thereby $t_0 \geqslant s_{\max}$ and $v \in \mathrm{In}^*_u(s_{\max}:t_0)$. This implies, by Lemma 8, that

$$r_u(t_0+1) = r_v(t_0+1) = t_0 + 1 - s_{max} = \min_{w \in V} r_w(t_0+1) \ .$$

Using Lemma 12 again, we get that for every node $w \in V$, $HO_w(t_0+1) = V$. Therefore the inequality (4) holds for all nodes in round t_0+1, and by the definition of t_0 this is the first round in which this inequality holds for all nodes. Hence all nodes fire simultaneously at the end of round t_0. ◀

The only operations in the algorithm $B_{c,T}$ that involve the node identities are performing the union and extracting the cardinalities of the sets HO_u. Since the decisions made by the algorithm are determined only by the cardinalities of the sets HO_u and not by the actual values of the identities in these sets, it is clear that the sequences of operations performed by each node in a specific run are independent of these values.

A close examination of the proof of Theorem 15, shows that each node actually computes the set V, and so its cardinality. As a byproduct, the algorithm $B_{c,T}$ thus solves the problem of counting the network size despite asynchronous starts in any model of dynamic graphs that are c connected with delay T, and in particular in the model of continuously strongly connected dynamic graphs. This should be compared with the impossibility result by Wattenhofer [24] which states that if passive nodes do not transmit any signal, then counting is impossible with asynchronous starts.

6 Bound on the Network Size and Randomization

In this section we show that if a polynomial bound N on the network size n is given, then randomization may reduce the message size in our firing squad algorithm $B_{c,T}$ without degrading its linear time complexity. Similarly to $B_{c,T}$, our randomized algorithm for the

firing squad problem actually estimates the size of the network, and thus as a byproduct, provides a solution to the approximate counting problem for the case of asynchronous starts. In this sense, it generalizes the randomized approximate counting algorithm of [16], which assumes that all nodes start simultaneously.

First observe that by Corollary 10, if we use N as an upper bound on the diameter of the network, then the A_N algorithm in Section 4 solves the firing squad problem within $O(N)$ rounds using messages of size $O(\log(N))$. When N is significantly larger than the network size n, this solution is thus not satisfactory regarding its time complexity.

For the sake of simplicity, we present our randomized firing squad algorithm in the case $c = T = 1$, i.e., for dynamic graphs that are continuously strongly connected, but the generalization to the case of c connectivity with delay T is straightforward. The algorithm, denoted $R_{N,\eta}$, depends on two parameters N and η, where N is a positive integer and η is any real number in $[\,0, 1/2\,)$. For this algorithm, it is assumed that the dynamic graph, and the start signals s_v, are managed by an *oblivious* adversary, which has no access to the outcomes of the random choices made by the algorithm.

The algorithm works as follows: upon becoming active, each node u generates ℓ independent random numbers $Y_u^{(1)}, \ldots, Y_u^{(\ell)}$, where ℓ depends on N and η, and the distribution of each $Y_u^{(i)}$ is exponential with rate 1. At each round, any active node u first broadcasts the smallest value of the $Y_v^{(i)}$'s it has heard of for each index $i \in \{1, \ldots, \ell\}$, and then computes from the minimum values it received so far an estimation n_u of the number of nodes it heard of. Node u fires when the value of its clock r_u is sufficiently large compared to n_u.

Using Cramér-Chernoff's bounds [2], we show that with high probability, the value of n_u at the end of round t provides a good approximation of the number of active nodes that u has heard of so far. This implies, via Lemma 12, that if $n_u < 2r_u/3$ then with high probability node u has heard of all other nodes (yielding the condition $n_u < 2r_u/3$ for node u to fire in line 14). As for the algorithm $B_{c,T}$, we conclude that with high probability, no node ever fires in a non-active run, and all nodes fire at the same round of any active run. More precisely, we choose $\ell = \lceil 243 \cdot (\ln 4N^2 - \ln \eta) \rceil$ to guarantee a final probability of at least $1 - \eta$ for these successful active and non-active runs.

The size of the messages used by the algorithm can be limited, at the price of higher storage capacity at the nodes, by using a rounded and range-restricted calculations as in [20]. Specifically, we round down each $Y_u^{(i)}$ to the next smaller integer power of 13/12, denoted $\overline{Y}_u^{(i)}$. Then the resulted approximate value $\overline{n_u}$ of n_u satisfies $n_u \leqslant \overline{n_u} \leqslant \frac{13}{12} n_u$, which guarantees that with high probability, $\overline{n_u}$ is also a good approximation of the number of active nodes that u has heard of so far.

By the definition of the exponential distribution, it is not hard to see that the random variables $Y_u^{(i)}$ are all within the range $[\eta/(4\ell N), \ln(4\ell N/\eta)]$ with high probability, namely

$$\Pr\left[\forall u \in V,\ \forall i,\ Y_u^{(i)} \in [\eta/(4\ell N), \ln(4\ell N/\eta)]\right] > 1 - \eta/2\ , \tag{5}$$

which allows us to ignore runs in which the randomized variables $Y_u^{(i)}$ are not in the above range. The number of distinct variables $\overline{Y}_u^{(i)}$ in that range is $O(\log(N\eta^{-1}))$, hence each such variable can be represented using $O\left(\log\log(N/\eta)\right)$ bits. This leads to messages length in $O\left(\log(N/\eta) \cdot \log\log(N/\eta)\right)$ bits.

We note, however, that the implied calculations require exponentially higher storage capacities: computing $\overline{n_u}$ (line 14 of algorithm $R_{N,\eta}$) must be done with the ℓ *exact* values of the variables $\overline{Y}_u^{(i)}$, and exact representation of numbers occurring in the implied calculations may require $\Omega(\ell N \eta^{-1})$ bits.

Algorithm 3: The randomized algorithm $R_{N,\eta}$, firing with continuous strong connectivity.

Initialization:

$r_u \in \mathbb{N}$, initially 0

$\overline{Y}_u = \left(\overline{Y}_u^{(1)}, \ldots, \overline{Y}_u^{(\ell)}\right) \in \mathbb{R}^\ell$ with $\ell = \left\lceil 243 \cdot (\ln 4N^2 - \ln \eta)\right\rceil$, initially rounded and range-restricted approximation of independent random numbers with exponential distribution of rate 1.

$n_u \in \mathbb{N}$, initially 0

In each round t do:

send $\langle r_u, \overline{Y}_u \rangle$ to all processes and receive one message from each in-neighbor

if at least one received message is null **then**

  $r_u \leftarrow 0$

else

  $r_u \leftarrow 1 + \min_{r \in M_u^{*(1)}}(r)$

end if

for $i = 1, \ldots, \ell$ **do**

  $\overline{Y}_u^{(i)} \leftarrow \min_{\overline{Y}^{(i)} \in M_u^{*(i+1)}} \left(\overline{Y}^{(i)}\right)$

end for

$\overline{n_u} \leftarrow \ell / \sum_{i=1}^{\ell} \overline{Y}_u^{(i)}$

if $\overline{n_u} < 2\, r_u/3$ **then**

  fire

end if

The correctness proof of the algorithm with the approximate random variables $\overline{Y}_u^{(i)}$ is valid for all runs in which the exact random variables are in the range (5), and this range restriction is violated with probability of at most $\eta/2$.

▶ **Theorem 16.** *In any dynamic graph that is continuously strongly connected and with at most N nodes, the algorithm $R_{N,\eta}$ solves the firing squad problem with probability at least $1-\eta$. Moreover, in any active run, with probability at least $1-\eta$, all nodes fire simultaneously in less than $2n$ rounds after the last nodes have become active.*

7 Conclusion and Further Research

In this paper we studied the firing squad problem in a network of an unknown number of nodes with full computational power, thus extending the original model which assumes that nodes are finite state machines. We focused on a natural extension of the problem in which start signals are left arbitrary, i.e., are no more supposed to be propagated by the nodes in the network.

We modeled the inter-node communication by a dynamic graph, and presented a tight relation between the solvability of the firing squad problem and the connectivity of the dynamic graph. Specifically, we introduced the notion of delayed connectivity, and showed that the firing squad problem is solvable if and only if the dynamic graph is connected with delay T, for some *given* constant T. Our solution uses messages of super-linear size, and we showed that additional information on the diameter or on the size of the network can substantially reduce the message size.

Combining our positive and negative results, we get that when nodes are infinite state machines, the firing squad problem is solvable for arbitrary timing of start signals if and only if it is solvable when restricted to diffusive start signals. An interesting question is whether this equivalence in terms of solvability is still valid in the original model of the firing squad problem, where nodes are finite state machines. It can be shown that this is the case when the topology is a line or a circuit, but it is not clear whether this holds for other topologies.

Possible extensions of this work involve other variations of the model of computation. For instance, it is interesting to determine under what conditions the firing squad problem is solvable in an anonymous network where nodes have limited storage capabilities and communicate through finite bandwith channels as in [15]. Our randomized algorithm provides an efficient Monte Carlo solution for this problem, in the case of a continuously strongly connected network and a polynomial upper bound on the size of the network. Another open question concerns the role of leaders in a dynamic network: does the existence of a leader could be useful for achieving or improving solutions to the firing squad problem?

References

1 Sebastian Abshoff, Markus Benter, Andreas Cord-Landwehr, Manuel Malatyali, and Friedhelm Meyer auf der Heide. Token dissemination in geometric dynamic networks. In *Proceedings of the 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics, ALGOSENSORS*, pages 22–34, 2013.

2 Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration inequalities. A nonasymptotic theory of independence*. Oxford University Press, Oxford, 2013.

3 James E. Burns and Nancy Lynch. The byzantine firing squad problem. *Advances in Computing Research*, 4:147–161, 1987.

4 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. In Hannes Frey, Xu Li, and Stefan Rührup, editors, *ADHOC-NOW*, volume 6811 of *Lecture Notes in Computer Science*, pages 346–359. Springer, 2011.

5 Bernadette Charron-Bost and André Schiper. The Heard-Of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 2009.

6 Bernard Chazelle. Natural algorithms and influence systems. *Communications of the ACM*, 55(12):101–110, 2012.

7 Brian A. Coan, Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. The distributed firing squad problem. In *ACM Symposium on Theory of Computing Conference, STOC'85*, pages 335–345, 1985.

8 Thiago Correa, Breno Gustavo, Lucas Lemos, and Amber Settle. An overview of recent solutions to and lower bounds for the firing synchronization problem. *arXiv preprint arXiv:1701.01045*, 2017.

9 Reinhard Diestel. *Graph Theory*. Springer-Verlag Berlin Heidelberg, 2017.

10 Danny Dolev, Ezra N. Hoch, and Yoram Moses. An optimal self-stabilizing firing squad. *SIAM Journal on Computing*, 41(2):415–435, 2012.

11 Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. Early stopping in Byzantine agreement. *jacm*, 37(4):720–741, 1990.

12 Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. 12(4):656–666, 1983.

13 Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a Byzantine environment: Crash failures. *Information and Computation*, 88(2):156–186, oct 1990.

14 Steven Finn. Resynch procedures and a fail-safe network protocol. *IEEE Transactions on Communications*, 27(6):840–845, 1979.

15 Julien M. Hendrickx, Alexander Olshevsky, and John N. Tsitsiklis. Distributed anonymous discrete function computation. *IEEE Trans. Automat. Contr.*, 56(10):2276–2289, 2011. doi:10.1109/TAC.2011.2163874.

16 Fabian Kuhn, Nancy A. Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on*

Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010, pages 513–522. ACM, 2010. doi:10.1145/1806689.1806760.

17 Fabian Kuhn, Yoram Moses, and Rotem Oshman. Coordinated consensus in dynamic networks. In *Proceedings of the 30th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–10. ACM, 2011.

18 Edward F. Moore. The firing squad synchronization problem. *Sequential Machines, Selected papers*, pages 213–214, 1964.

19 F. R. Moore and G. G. Langdon. A generalized firing squad problem. *Information and Control*, 12(3):212–220, 1968.

20 Rotem Oshman. *Distributed Computation in Wireless and Dynamic Networks*. PhD thesis, Massachusetts Institute of Technology, 2012.

21 Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. 27(2):228–234, 1980.

22 Nicola Santoro. Time to change: On distributed computing in dynamic networks (keynote). In *19th International Conference on Principles of Distributed Systems, OPODIS 2015, December 14-17, 2015, Rennes, France*, pages 3:1–3:14, 2015.

23 T. K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2):80–94, 1987.

24 Roger Wattenhofer. Principles of distributed computing. Unpublished, 2014.

Small-depth Multilinear Formula Lower Bounds for Iterated Matrix Multiplication, with Applications

Suryajith Chillara
Chennai Mathematical Institute, India
suryajith@cmi.ac.in

Nutan Limaye
Department of CSE, IIT Bombay, India
nutan@cse.iitb.ac.in

Srikanth Srinivasan
Department of Mathematics, IIT Bombay, India
srikanth@math.iitb.ac.in

Abstract

The complexity of Iterated Matrix Multiplication is a central theme in Computational Complexity theory, as the problem is closely related to the problem of separating various complexity classes within P. In this paper, we study the algebraic formula complexity of multiplying d many 2×2 matrices, denoted IMM_d, and show that the well-known divide-and-conquer algorithm cannot be significantly improved at any depth, as long as the formulas are multilinear.

Formally, for each depth $\Delta \leq \log d$, we show that any product-depth Δ multilinear formula for IMM_d must have size $\exp(\Omega(\Delta d^{1/\Delta}))$. It also follows from this that any multilinear circuit of product-depth Δ for the same polynomial of the above form must have a size of $\exp(\Omega(d^{1/\Delta}))$. In particular, any polynomial-sized multilinear formula for IMM_d must have depth $\Omega(\log d)$, and any polynomial-sized multilinear circuit for IMM_d must have depth $\Omega(\log d/\log\log d)$. Both these bounds are tight up to constant factors.

Our lower bound has the following consequences for multilinear formula complexity.

1. **Depth-reduction**: A well-known result of Brent (JACM 1974) implies that any formula of size s can be converted to one of size $s^{O(1)}$ and depth $O(\log s)$; further, this reduction continues to hold for multilinear formulas. On the other hand, our lower bound implies that any depth-reduction in the multilinear setting cannot reduce the depth to $o(\log s)$ without a superpolynomial blow-up in size.
2. **Separations from general formulas**: Shpilka and Yehudayoff (FnTTCS 2010) asked whether general formulas can be more efficient than multilinear formulas for computing multilinear polynomials. Our result, along with a non-trivial upper bound for IMM_d implied by a result of Gupta, Kamath, Kayal and Saptharishi (SICOMP 2016), shows that for any size s and product-depth $\Delta = o(\log s)$, general formulas of size s and product-depth Δ cannot be converted to multilinear formulas of size $s^{O(1)}$ and product-depth Δ, when the underlying field has characteristic zero.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases Algebraic Circuit Complexity, Multilinear Formulas, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.21

Related Version A full version of the paper is available at [4], `https://eccc.weizmann.ac.il/report/2017/156/`.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 21; pp. 21:1–21:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Algebraic Complexity theory is the study of the complexity of those computational problems that can be phrased as computing a multivariate polynomial $f(x_1, \ldots, x_N) \in \mathbb{F}[x_1, \ldots, x_N]$ over elements $x_1, \ldots, x_N \in \mathbb{F}$. Many central algorithmic problems such as the Determinant, Permanent, Matrix product etc. can be cast in this framework. The natural computational models that we consider in this setting are models such as *Algebraic circuits*, *Algebraic Branching Programs* (ABPs), and *Algebraic formulas* (or just formulas), all of which use the natural algebraic operations of $\mathbb{F}[x_1, \ldots, x_N]$ to compute the polynomial f. These models have by now been the subject of a large body of work with many interesting upper bounds (i.e. circuit constructions) as well as lower bounds (i.e. impossibility results). (See, e.g. the surveys [23, 22] for an overview of many of these results.)

Despite this, many fundamental questions remain unresolved. An important example of such a question is that of proving lower bounds on the size of formulas for the *Iterated Matrix Multiplication* problem, which is defined as follows. Given d $n \times n$ matrices $M_1, \ldots, M_d$, we are required to compute (an entry of) the product $M_1 \cdots M_d$; we refer to this problem as $\mathrm{IMM}_{n,d}$. Proving superpolynomial lower bounds on the size of formulas for this problem is equivalent to separating the power of polynomial-sized ABPs from polynomial-sized formulas, which is the algebraic analogue of separating the Boolean complexity classes NL and NC^1.

A standard divide-and-conquer algorithm yields the best-known formulas for $\mathrm{IMM}_{n,d}$. More precisely, for any $\Delta \leq \log d$, this approach yields a formula of product-depth[1] Δ and size $n^{O(\Delta d^{1/\Delta})}$ for $\mathrm{IMM}_{n,d}$ and choosing $\Delta = \log d$ yields the current best formula upper bound of $n^{O(\log d)}$, which has not been improved in quite some time. On the other hand, separating the power of ABPs and formulas is equivalent to showing that $\mathrm{IMM}_{n,d}$ does not have formulas of size $\mathrm{poly}(nd)$.

The Iterated Matrix Multiplication problem has many nice features that render its complexity an interesting object to study. For one, it is the algebraic analogue of the Boolean reachability problem, and thus any improved formula upper bounds for $\mathrm{IMM}_{n,d}$ could lead to improved Boolean circuit upper bounds for the reachability problem, which would resolve a long-standing open problem in that area. For another, this problem has strong self-reducibility properties, which imply that improving on the simple divide-and-conquer approach to obtain formulas of size $n^{o(\log d)}$ for any d would lead to improved upper bounds for all $D > d$; this implies that the lower-degree variant is no easier than the higher-degree version of the problem, which can be very useful (e.g. for homogenization [16]). Finally, the connection to the Reachability problem imbues $\mathrm{IMM}_{n,d}$ with a rich combinatorial structure via its graph theoretic interpretation, which has been used extensively in lower bounds for depth-4 arithmetic circuits [6, 9, 12, 10, 11].

We study the formula complexity of this problem in the *multilinear* setting, which restricts the underlying formulas to only compute multilinear polynomials at intermediate stages of computation. Starting with the breakthrough work of Raz [15], many lower bounds have been proved for multilinear models of computation [18, 19, 17, 5]. Further, it is known by a result of Dvir, Malod, Perifel and Yehudayoff [5] that multilinear ABPs are in fact superpolynomially more powerful than multilinear formulas. Unfortunately, however, this does not imply any non-trivial lower bound for Iterated Matrix Multiplication (see the Related Work section

[1] The *product-depth* of an arithmetic circuit or formula is the maximum number of product gates on a path from output to input. If the product-depth of a circuit or formula is Δ, then its depth can be assumed to be at least $2\Delta - 1$ and at most $2\Delta + 1$.

below), and as far as we know, it could well be the case that there are multilinear formulas that beat the divide-and-conquer approach in computing this polynomial.

Here, we are able to show that this is not the case for the problem of multiplying 2×2 matrices (and by extension $c \times c$ matrices for any constant c) at any product-depth. Our main theorem is the following (stated more formally as Theorem 9 later).

▶ **Theorem 1.** *For $\Delta \leq \log d$, any product-depth Δ multilinear formula that computes $\mathrm{IMM}_{2,d}$ must have size $2^{\Omega(\Delta d^{1/\Delta})}$.*

This lower bound strengthens a result of Nisan and Wigderson [13] who prove a similar lower bound in the more restricted *set-multilinear* setting.

Our result is also qualitatively different from the previous lower bounds for multilinear formulas since $\mathrm{IMM}_{2,d}$ does in fact have polynomial-sized formulas of product-depth $O(\log d)$ (via the divide-and-conquer approach), whereas we show a superpolynomial lower bound for product-depth $o(\log d)$. This observation leads to interesting consequences for multilinear formula complexity in general, which we now describe.

- Depth Reduction: An important theme in Circuit complexity is the interplay between the size of a formula or circuit and its depth [3, 24, 26, 1, 25]. In the context of algebraic formulas, a result of Brent [3] says that any formula of size s can be converted into another of size $s^{O(1)}$ and depth $O(\log s)$. Further, the proof of this result also yields the same statement for multilinear formulas.
  Can the result of Brent be improved? Theorem 1 implies that the answer is no in the multilinear setting (Corollary 12). More precisely, since the $\mathrm{IMM}_{2,d}$ polynomial (over $O(d)$ variables) has formulas of size $\mathrm{poly}(d)$ and depth $O(\log d)$ but no formulas of size $d^{O(1)}$ and depth $o(\log d)$ (by Theorem 1), we see that any multilinear depth-reduction procedure that reduces the depth of a size-s formula to $o(\log s)$ must incur a superpolynomial blow-up in size. This strengthens a result of Raz and Yehudayoff [19], whose results imply that any depth-reduction of multilinear formulas to depth $o(\sqrt{\log s}/\log\log s)$ should incur a superpolynomial blow-up in size. It is also an analogue in the algebraic setting of some recent results proved for Boolean circuits [20, 21].
- Multilinear vs. general formulas: Shpilka and Yehudayoff [23] ask the question of whether general formulas can be more efficient at computing multilinear polynomials than multilinear formulas. This is an important question, since we have techniques for proving lower bounds for multilinear formulas, whereas the same question for general formulas (or even depth-3 formulas over large fields) remains wide open.
  We are able to make progress towards this question here by showing a separation between the two models for small depths when the underlying field has characteristic zero (Corollary 13). We do this by using Theorem 1 in conjunction with a (non-multilinear) formula *upper bound* for $\mathrm{IMM}_{2,d}$ over fields of characteristic zero due to Gupta et al. [7]. In particular, the result of Gupta et al. [7] implies that for any depth Δ, the polynomial $\mathrm{IMM}_{2,d}$ has formulas of product depth Δ and size $2^{O(\Delta d^{1/2\Delta})}$, which is considerably smaller than our lower bound in the multilinear case for small Δ. From this, it follows that for any size parameter s and product-depth $\Delta = o(\log s)$, general formulas of size s and product-depth Δ cannot be converted to multilinear formulas of size $s^{O(1)}$ and product-depth Δ. Improving our result to allow for $\Delta = O(\log s)$ would resolve the question entirely.

Related Work. The multilinear formula model has been the focus of a large body of work on Algebraic circuit lower bounds. Nisan and Wigderson [13] proved some of the early

results in this model by showing size lower bounds for small-depth *set-multilinear*[2] circuits computing $\mathrm{IMM}_{2,d}$. They showed that any product-depth Δ circuit for $\mathrm{IMM}_{2,d}$ must have a size of $2^{\Omega(d^{1/\Delta})}$ matching the upper bound from the divide-and-conquer algorithm for $\Delta = o(\log d/\log\log d)$. Our lower bounds for multilinear formulas imply similar lower bounds for multilinear circuits of product-depth Δ.

Raz [15] proved the first superpolynomial lower bound for multilinear formulas by showing an $n^{\Omega(\log n)}$ lower bound for the $n \times n$ Determinant and Permanent polynomials. This was further strengthened by the results of Raz [14] and Raz and Yehudayoff [18] to a similar lower bound for an explicit polynomial family that has polynomial-sized multilinear *circuits*. In particular, these results show the tightness of the depth-reduction procedure for algebraic *circuits* in the multilinear setting [26, 18].

Similar polynomial families were also used in the work of Raz and Yehudayoff [19] to prove *exponential* lower bounds for multilinear constant-depth circuits. By proving a tight lower bound for depth-Δ circuits computing an explicit polynomial (similar to the construction of Raz [14]), Raz and Yehudayoff [19] showed superpolynomial separations between multilinear circuits of different depths.

In particular, the result of Raz and Yehudayoff [19] implies that the polynomial families of [14, 18], which have formulas of size $n^{O(\log n)}$, cannot be computed by formulas of size less than some $s(n) = n^{\omega(\log n)}$ if the product-depth $\Delta = o(\log n/\log\log n)$. This yields the superpolynomial separation between formulas of size s and depth $o(\sqrt{\log s}/\log\log s)$ alluded to above. Unfortunately, these polynomials also have nearly optimal formulas of depth just $O(\log n) = O(\sqrt{\log s})$, so they cannot be used to obtain the optimal size s vs depth $o(\log s)$ separation we obtain here.

Dvir et al. [5] showed that there is an explicit polynomial on n variables that has multilinear ABPs of size $\mathrm{poly}(n)$ but no multilinear formulas of size less than $n^{\Omega(\log n)}$. One might hope that this yields a superpolynomial lower bound for multilinear formulas computing $\mathrm{IMM}_{N,d}$ for some N, d but this unfortunately does not seem to be the case. The reason for this is that while any polynomial f on n variables that has an ABP of size $\mathrm{poly}(n)$ can be reduced via variable substitutions to $\mathrm{IMM}_{N,d}$ for $N, d = n^{O(1)}$, this reduction might substitute different variables in the $\mathrm{IMM}_{N,d}$ polynomial by the same variable x of f and in the process destroy multilinearity.

Gupta et al. [7] showed the surprising result that general (i.e. non-multilinear) formulas of depth-3 can beat the divide-and-conquer approach for computing $\mathrm{IMM}_{n,d}$, when the underlying field has characteristic zero. Their result implies that, in this setting, $\mathrm{IMM}_{n,d}$ has product-depth 1 formulas of size $n^{O(\sqrt{d})}$, as opposed to the $n^{O(d)}$-sized formula that is obtained from the traditional divide-and-conquer approach. Using the self-reduction properties of $\mathrm{IMM}_{n,d}$, this can be easily seen to imply the existence of $n^{O(\Delta d^{1/2\Delta})}$-sized formulas of product-depth Δ. This construction uses the fact that the formulas are allowed to be non-multilinear. Our result shows that this cannot be avoided.

[2] Set-multilinear circuits are further restrictions of multilinear circuits. A set-multilinear circuit for $\mathrm{IMM}_{n,d}$ is defined by the property that each intermediate polynomial computed must be a linear combination of monomials that contain exactly one variable from each matrix M_i ($i \in S$), for some choice of $S \subseteq [d]$.

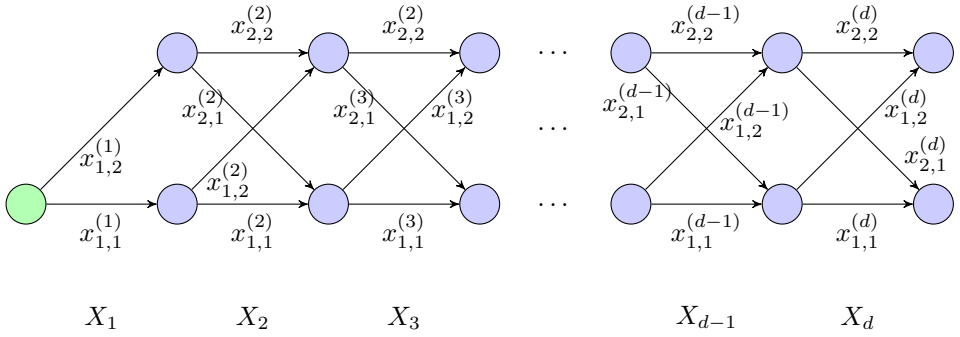


Figure 1 The directed acyclic graph G_d that defines the polynomial IMM_d with its labeling.

2 Preliminaries

2.1 Basic setup

Unless otherwise stated, let $\mathbb{F}$ be an arbitrary field. Let $d \in \mathbb{N}$ be a growing integer parameter. We define $X^{(1)}, \ldots, X^{(d)}$ to be disjoint sets of variables where each $X^{(i)} = \{x^{(i)}_{j,k} \mid j,k \in [2]\}$ is a set of four variables that we think of forming a 2×2 matrix. Let $X = \bigcup_{i\in[d]} X^{(i)}$.

A polynomial $P \in \mathbb{F}[X]$ is called *multilinear* if the degree of P in each variable $x \in X$ is at most 1. We define the multilinear polynomial $\mathrm{IMM}_d \in \mathbb{F}[X]$ as follows. Consider the matrices $M^{(1)}, \ldots, M^{(d)}$ where the entries of $M^{(i)}$ are the variables of $X^{(i)}$ arranged in the obvious way. Define the matrix $M = M^{(1)} \cdots M^{(d)}$; the entries of M are multilinear polynomials over the variables in X. We define $\mathrm{IMM}_d = M(1,1) + M(1,2)$, i.e. the sum of the $(1,1)$th and $(1,2)$th entries of M. Note, in particular, that the polynomial IMM_d does not depend on the variables $x^{(1)}_{2,1}$ and $x^{(1)}_{2,2}$.

This is a slight variant of the Iterated Matrix Multiplication polynomial seen in the literature, as it is usually defined to be either the matrix entry $M(1,1)$ or the trace $M(1,1) + M(2,2)$. Our results can easily be seen to hold for these variants, but we deal with the definition above for some technical simplicity.

Another standard way of defining the polynomial IMM_d is via graphs. Define the edge-labelled directed acyclic graph $G_d = (V, E, \lambda)$ as follows: the vertex set V is defined to be the disjoint union of vertex sets $V^{(0)}, \ldots, V^{(d)}$ where $V^{(i)} = \{v^{(i)}_1, v^{(i)}_2\}$. The edge set E is the set of all possible edges from some set $V^{(i)}$ to $V^{(i+1)}$ (for all $i < d$). The labelling function $\lambda : E \to X$ is defined by $\lambda((v^{(i)}_j, v^{(i+1)}_k)) = x^{(i+1)}_{j,k}$. See Figure 1 for a depiction of this graph.

Given a path π in the graph G_d, $\lambda(\pi)$ is defined to be the product of all labels of edges in π. In this notation, IMM_d can be seen to be the following.

$$\mathrm{IMM}_d = \sum_{\substack{\text{paths } \pi \text{ from } v^{(0)}_1 \\ \text{to } v^{(d)}_1 \text{ or } v^{(d)}_2}} \lambda(\pi) = \sum_{\pi_1,\ldots,\pi_d \in \{1,2\}} x^{(1)}_{1,\pi_1} x^{(2)}_{\pi_1,\pi_2} \cdots x^{(d)}_{\pi_{d-1},\pi_d} \tag{1}$$

2.2 Multilinear formulas and circuits

We refer the reader to the standard resources (e.g. [23, 22]) for basic definitions related to algebraic circuits and formulas. Having said that, we do make a few remarks.

- All the gates in our formulas and circuits will be allowed to have *unbounded* fan-in.

- The size of a formula or circuit will refer to the number of gates (including input gates) in it, and depth of the formula or circuit will refer to the number of gates on the longest path from an input gate to output gate.
- Further, the *product-depth* of the formula or circuit (as in [18]) will refer to the maximum number of product gates on a path from an input gate to the output gate. Note that the product depth of a formula or circuit can be assumed to be within a factor of two of the overall depth (by collapsing sum gates if necessary).

Multilinear circuits and formulas. An algebraic formula F (resp. circuit C) computing a polynomial from $\mathbb{F}[X]$ is said to be *multilinear* if each gate in the formula (resp. circuit) computes a multilinear polynomial [3]. Moreover, a formula F is said to be *syntactic multilinear* if for each multiplication gate Φ of F with children $\Psi_1, \ldots, \Psi_t$, we have $\text{Supp}(\Psi_i) \cap \text{Supp}(\Psi_j) = \emptyset$ for each $i \neq j$, where $\text{Supp}(\Phi)$ denotes the set of variables that appear in the subformula rooted at Φ. Finally, for $\Delta \geq 1$, we say that a multilinear formula (resp. circuit) is a $(\Sigma\Pi)^{\Delta}\Sigma$ formula (resp. circuit) if the output gate is a sum gate and along any path, the sum and product gates alternate, with each product gate appearing exactly Δ times and the bottom gate being a sum gate. We can define $(\Sigma\Pi)^{\Delta}, \Sigma\Pi\Sigma, \Sigma\Pi\Sigma\Pi$ formulas and circuits similarly.

For a gate Φ in a syntactically multilinear formula, we define a set of variables $\text{Vars}(\Phi)$ in a top-down fashion as follows.

▶ **Definition 2.** Let C be a syntactically multilinear formula computing a polynomial on the variable set X. For the output gate Φ, we define $\text{Vars}(\Phi) = X$. If Φ is a sum gate with children $\Psi_1, \ldots, \Psi_k$ and $\text{Vars}(\Phi) = S \subseteq X$, then for each $1 \leq i \leq k$, $\text{Vars}(\Psi_i) = S$. If Φ is a product gate with children $\Psi_1, \ldots \Psi_k$ and $\text{Vars}(\Phi) = S \subseteq X$, then $\text{Vars}(\Psi_i) = \text{Supp}(\Psi_i)$ for $1 \leq i \leq k-1$ and $\text{Vars}(\Psi_k) = S \setminus \left(\cup_{i=1}^{k-1} \text{Vars}(\Psi_i)\right)$.

It is easy to see that $\text{Vars}(\cdot)$ satisfies the properties listed in the following proposition.

▶ **Proposition 3.** *For each gate Φ in a syntactically multilinear formula C, let $\text{Vars}(\Phi)$ be defined as in Definition 2 above.*

1. *For any gate Φ in C, $\text{Supp}(\Phi) \subseteq \text{Vars}(\Phi)$.*
2. *If Φ is a sum gate, with children $\Psi_1, \Psi_2, \ldots, \Psi_k$, then $\forall i \in [k]$, $\text{Vars}(\Psi_i) = \text{Vars}(\Phi)$.*
3. *If Φ is a product gate, with children $\Psi_1, \Psi_2, \ldots, \Psi_k$, then $\text{Vars}(\Phi) = \cup_{i=1}^{k} \text{Vars}(\Psi_i)$ and the sets $\text{Vars}(\Psi_i)$ $(i \in [k])$ are pairwise disjoint.*

We will use the following structural results that convert general multilinear circuits (resp. formulas) to $(\Sigma\Pi)^{\Delta}\Sigma$ circuits (resp. formulas).

▶ **Lemma 4** (Raz and Yehudayoff [19], Claims 2.3 and 2.4)**.** *For any multilinear formula F of product depth at most Δ and size at most s, there is a syntactic multilinear $(\Sigma\Pi)^{\Delta}\Sigma$ formula F' of size at most $(\Delta+1)^2 \cdot s$ computing the same polynomial as F.*

▶ **Lemma 5** (Raz and Yehudayoff [19], Lemma 2.1)**.** *For any multilinear circuit C of product depth at most Δ and size at most s, there is a syntactic multilinear $(\Sigma\Pi)^{\Delta}\Sigma$ formula F of size at most $(\Delta+1)^2 \cdot s^{2\Delta+1}$ computing the same polynomial as C.*

We will also need the following structural result.

[3] It is important to note that any multilinear polynomial can be computed by a non-multilinear formula (resp. circuit) as well.

▶ **Lemma 6** (Raz, Shpilka and Yehudayoff [17], Claim 5.6). *Let F be a syntactic multilinear formula computing a polynomial f and let Φ be any gate in F computing a polynomial g. Then f can be written as $f = Ag + B$, where $A \in \mathbb{F}[X \setminus \mathrm{Vars}(\Phi)]$, $B \in \mathbb{F}[X]$ and B is computed by replacing Φ with a 0 in F.*

A standard divide-and-conquer approach (see [2, Proposition 3.10]) yields the best-known multilinear formulas/circuits for IMM_d for all depths. (A proof sketch is presented in [4].)

▶ **Lemma 7.** *For each $\Delta \leq \log d$,[4] IMM_d is computed by a syntactic multilinear $(\Sigma\Pi)^{\Delta}$ circuit C_Δ of size at most $d^{O(1)} \cdot 2^{O(d^{1/\Delta})}$ and a syntactic multilinear $(\Sigma\Pi)^{\Delta}$ formula F_Δ of size at most $2^{O(\Delta d^{1/\Delta})}$.*

We will show that the above bounds are nearly tight in the multilinear setting. If we remove the multilinear restriction on $(\Sigma\Pi)^{\Delta}\Sigma$ formulas computing IMM_d, we can get better upper bounds, as long as the underlying field has characteristic zero.

▶ **Lemma 8** (follows from [7]). *Let $\mathbb{F}$ be a field of characteristic zero. For each $\Delta \leq \log d$, IMM_d has a $(\Sigma\Pi)^{\Delta}\Sigma$ formula F_Δ of size at most $2^{O(\Delta d^{1/(2\Delta)})}$.*

A proof sketch for the above lemma is presented in the full version [4].

3 Lower bounds for multilinear formulas and circuits computing IMM_d

The main theorem of this section is the following lower bound.

▶ **Theorem 9.** *Let $d \geq 1$ be a growing parameter and fix any $\Delta \leq \log d$. Any syntactic multilinear $(\Sigma\Pi)^{\Delta}\Sigma$ formula for IMM_d must have a size of $2^{\Omega(\Delta d^{1/\Delta})}$.*

By applying Theorem 9 and Lemmas 4, 5, we get the following (immediate) corollaries.

▶ **Corollary 10.** *Let $d \geq 1$ be a growing parameter and fix any $\Delta \leq \log d/\log\log d$. Any multilinear circuit of product-depth at most Δ for IMM_d must have a size of $2^{\Omega(d^{1/\Delta})}$. In particular, any polynomial-sized multilinear circuit for IMM_d must have product-depth $\Omega(\log d/\log\log d)$.*

▶ **Corollary 11.** *Let $d \geq 1$ be a growing parameter and fix any $\Delta \leq \log d$. Any multilinear formula of product depth at most Δ for IMM_d must have size $2^{\Omega(\Delta d^{1/\Delta})}$. In particular, any polynomial-sized multilinear formula for IMM_d must have product-depth $\Omega(\log d)$.*

As the product-depth of a formula is at most its depth, Lemma 7, Corollary 11 imply:

▶ **Corollary 12** (Tightness of Brent's depth-reduction for multilinear formulas). *For each $d \geq 1$, there is an explicit polynomial F_d on $O(d)$ variables such that F_d has a multilinear formula of size $d^{O(1)}$, but any multilinear formula of depth $o(\log d)$ for F_d must have a size of $d^{\omega(1)}$.*

Choosing parameters carefully, we also obtain the following.

▶ **Corollary 13** (Separation of multilinear formulas and general formulas over zero characteristic). *Let $\mathbb{F}$ be a field of characteristic zero. Let $s \in \mathbb{N}$ be any growing parameter and $\Delta \in \mathbb{N}$ be such that $\Delta \leq o(\log s)$. There is an explicit multilinear polynomial $F_{s,\Delta}$ that has a $(\Sigma\Pi)^{\Delta}\Sigma$ formula of size s, but any $(\Sigma\Pi)^{\Delta}\Sigma$ multilinear formula for $F_{s,\Delta}$ must have a size of $s^{\omega(1)}$.*

[4] All our logarithms will be to base 2.

Proof. We choose the polynomial $F_{s,\Delta}$ to be IMM_d for suitable d and then simply apply Corollary 11 and Lemma 8 to obtain the result. Details follow.

Say $\Delta = \frac{\log s}{f(s)}$ for some $f(s) = \omega(1)$. By Lemma 8, for any d, IMM_d has a product-depth Δ formula of size $s(d,\Delta) = 2^{O(\Delta d^{1/2\Delta})}$; we choose d so that $s(d,\Delta) = s$. It can be checked that for $d = \Theta(f(s))^{2\Delta}$, this is indeed the case.

Having chosen d as above, we define $F_{s,\Delta} = \mathrm{IMM}_d$. Clearly, $F_{s,\Delta}$ has a (non-multilinear) formula of product-depth Δ and size at most s. On the other hand, by Theorem 9, any multilinear product-depth Δ formula for IMM_d must have size at least $2^{\Omega(\Delta d^{1/\Delta})} = s^{\Omega(d^{1/2\Delta})} = s^{\Omega(f(s))} = s^{\omega(1)}$, which proves the claim.

It can also be proved similarly that for d as chosen above, IMM_d in fact has no multilinear formulas of size $s^{O(1)}$ and product-depth up to $(2-\varepsilon)\Delta$ for any absolute constant ε. ◀

4 Proof of Theorem 9

Our proof follows a two-step argument as in [15, 19] (see the exposition in [23, Section 3.6]).

Step 1 – The product lemma. The first step is a "product-lemma" for multilinear formulas. Formally, define a polynomial $f \in \mathbb{F}[X]$ to be a *t-product polynomial* if we can write f as $f_1 \cdots f_t$, where we can find a partition of X into non-empty sets $X_1^f, \ldots, X_t^f$ such that f_i is a multilinear polynomial from $\mathbb{F}[X_i^f]$.[5] We say that X_i^f is the set *ascribed* to f_i in the t-product polynomial f. We use $\mathrm{Vars}(f_i)$ (with a slight abuse of notation)[6] to denote X_i^f. We drop f from the superscript if f is clear from the context.

We define $f \in \mathbb{F}[X]$ to be *r-simple* if $f = L_1 \cdots L_{r'} \cdot G$, where $r' \leq r$, is an $(r'+1)$-product polynomial where $L_1, \ldots, L_{r'}$ are polynomials of degree at most 1, the sets $X_1^f, \ldots, X_{r'}^f$ ascribed to these linear polynomials satisfy $\left|\bigcup_{i \leq r'} X_i^f\right| \geq 400r$. We prove the following.

▶ **Lemma 14.** *Let $\Delta \leq \log d$. Assume that $f \in \mathbb{F}[X]$ can be computed by a syntactic multilinear $(\Sigma\Pi)^{\Delta}\Sigma$ formula F of size at most s. Then, f is the sum of at most s many t-product polynomials and at most s many t-simple polynomials for $t = \Omega(\Delta d^{1/\Delta})$.*

While our proof (presented in the full version [4]) of this lemma is motivated by earlier work [23, 8, 19], we give slightly better parameters, which is crucial for proving tight lower bounds for formulas. In particular, [19, Claim 5.5] yields the above but with $t = \Omega(d^{1/\Delta})$.

Step 2 – Rank measure and the hard polynomial. The second step is to show that any such decomposition for IMM_d must have many terms. Our proof of this step is inspired by the proof of the multilinear formula lower bound of Raz [15] for the determinant and also the slightly weaker lower bound of Nisan and Wigderson [13] for IMM_d in the *set-multilinear* case. Following [15], we define a suitable *random restriction* ρ, of the IMM_d polynomial by assigning variables from the set X to $Y \cup Z \cup \{0,1\}$, where Y and Z are disjoint sets of new variables. As the restriction sets distinct variables in X to distinct variables in $Y \cup Z$ or constants, it preserves multilinearity.

Having performed the restriction, we consider the *partial derivative matrix* of the restricted polynomial, which is defined as follows. Let $g \in \mathbb{F}[Y \cup Z]$ be a multilinear polynomial. Define

[5] Note that we do not need f_i to depend non-trivially on all (or any) of the variables in X_i^f.

[6] $\mathrm{Vars}(\cdot)$ is used to describe variables ascribed to gates in a circuit as well as to denote variables ascribed to polynomials.

Algorithm 1 Sampling algorithm $\mathcal{S}$.

Choose π uniformly at random from $\{1,2\}^d$. Define $\pi(0) = 1$.
Choose a uniformly at random from $\{0,1\}^d$. Let $A = \{i \mid a_i = 1\}$.
for $i \in [d]$ **do**
  Let $b_i = 0$ if $\pi(i-1) = \pi(i)$ and 1 if $\pi(i-1) \neq \pi(i)$.
end for
for $i = 1$ to d **do**
  if $i \notin A$ **then**
    Choose $\rho|_{X^{(i)}}$ such that $M^{(i)}$ is I if $b_i = 0$ and E if $b_i = 1$. (In particular, all variables are set to constants from $\{0,1\}$.)
  else if $i \in A$ and i is the jth smallest element of A for odd j **then**
    Fix

$$\rho(x^{(i)}_{u,v}) = \begin{cases} y_{\lceil j/2 \rceil} & \text{if } u = \pi(i-1) \text{ and } v = \pi(i), \\ 1 & \text{if } u = \pi(i-1) \text{ and } v = \overline{\pi(i)}, \\ 0 & \text{otherwise.} \end{cases}$$

  else
    Now, $i \in A$ and i is the jth smallest element of A for even j. We fix

$$\rho(x^{(i)}_{u,v}) = \begin{cases} z_{j/2} & \text{if } u = \pi(i-1) \text{ and } v = \pi(i), \\ 1 & \text{if } u = \overline{\pi(i-1)} \text{ and } v = \pi(i), \\ 0 & \text{otherwise.} \end{cases}$$

  end if
end for

the $2^{|Y|} \times 2^{|Z|}$ matrix $M_{(Y,Z)}(g)$ such that rows and columns are labelled by distinct multilinear monomials in Y and Z respectively and the (m_1, m_2)th entry of $M_{(Y,Z)}(g)$ is the coefficient of the monomial $m_1 \cdot m_2$ in g.

Our restriction is defined to have the following two properties.

1. The rank of $M_{(Y,Z)}(g)$ is equal to its maximum possible value (i.e. $\min\{2^{|Y|}, 2^{|Z|}\}$) with probability 1 where g is the restricted version of IMM_d.
2. On the other hand, let f be either a t-product or a t-simple polynomial, and let f' denote its restriction under ρ. Then, the rank of $M_{(Y,Z)}(f')$ is small with high probability.

Now, if IMM_d has a $(\Sigma\Pi)^{\Delta}\Sigma$ formula F of small size, then it is a sum of a small number of t-product and t-simple polynomials by Lemma 14 and hence by a union bound, we will be able to find a restriction under which the partial derivative matrices of each of these polynomials have a small rank. By the subadditivity of rank, this will imply that $M_{(Y,Z)}(g)$ will itself have low rank, contradicting the first property of our restriction.

To make the above precise, we first define our restrictions. Let $\tilde{Y} = \{y_1, \ldots, y_d\}$ and $\tilde{Z} = \{z_1, \ldots, z_d\}$ be two disjoint sets of variables. A restriction ρ is a function mapping the set X to $\tilde{Y} \cup \tilde{Z} \cup \{0,1\}$. We consider the following process for sampling a random restriction.

Recall that $M^{(i)}$ is the 2×2 matrix whose (u,v)th entry is $x^{(i)}_{u,v}$. Let I and E denote the standard 2×2 identity matrix and the 2×2 flip permutation matrix respectively. For $a \in \{1,2\}$, we use $\overline{a}$ to denote the other element of the set.

We give a procedure $\mathcal{S}$ for sampling a random restriction $\rho : X \to \tilde{Y} \cup \tilde{Z} \cup \{0,1\}$ in Algorithm 1. Based on the output ρ of $\mathcal{S}$, we define the (random) sets $Y = \tilde{Y} \cap \mathrm{Img}(\rho)$ and $Z = \tilde{Z} \cap \mathrm{Img}(\rho)$. Let $m = m(\rho) = \min\{|Y|, |Z|\}$.

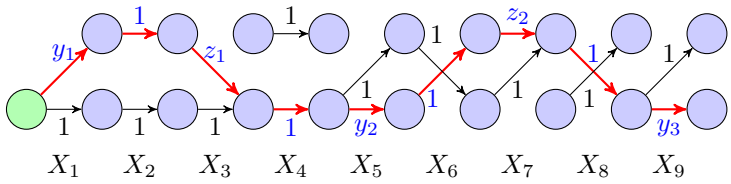


Figure 2 Effect of ρ on IMM_9 when the sampling algorithm $\mathcal{S}$ yields $\pi = (2,2,1,1,1,2,2,1,1)$ and $a = (1,0,1,0,1,0,1,0,1)$. Thus, $\mathrm{IMM}_9|_\rho$ in this case yields us $(1+y_1z_1)(1+y_2z_2)(1+y_3)$.

▶ **Observation 15.** *The restriction ρ satisfies the following.*

1. $|Y| = \lceil |A|/2 \rceil$ *and* $|Z| = \lfloor |A|/2 \rfloor$. *Hence,* $|Z| \leq |Y| \leq |Z|+1$ *and* $m = |Z|$.
2. *Distinct variables in X cannot be mapped to the same variable in $Y \cup Z$.*
3. *Only the variables of the form $x^{(i)}_{\pi(i-1),\pi(i)}$ can be set to variables in $Y \cup Z$ by ρ. The rest are set to constants.*

Note that b is distributed uniformly over $\{0,1\}^d$. Given a polynomial $f \in \mathbb{F}[X]$, the restriction ρ yields a natural polynomial $f|_\rho \in \mathbb{F}[Y \cup Z]$ by substitution. Note, moreover, that if f is multilinear then so is $f|_\rho$ since distinct variables in X cannot be mapped to the same variable in $Y \cup Z$ (Observation 15).

▶ **Lemma 16.** *Let us assume that ρ is sampled as above. Then we have the following:*

1. $\mathrm{rank}(M_{(Y,Z)}(\mathrm{IMM}_d|_\rho)) = 2^m$ *with probability* 1.
2. *If $f \in \mathbb{F}[X]$ is any t-product polynomial, then for some absolute constant $\varepsilon > 0$,,* $\Pr[\mathrm{rank}(M_{(Y,Z)}(f|_\rho)) \geq 2^{m-\varepsilon t}] \leq \frac{1}{2^{\Omega(t)}}$.
3. *If $f \in \mathbb{F}[X]$ is any r-simple polynomial, then for some absolute constant $\delta > 0$,* $\Pr[\mathrm{rank}(M_{(Y,Z)}(f|_\rho)) \geq 2^{m-\delta r}] \leq \frac{1}{2^{\Omega(r)}}$.

Given Lemmas 14 and 16, we can finish the proof of Theorem 9 as follows.

Proof of Theorem 9 assuming Lemma 16. Assume that IMM_d is computed by a syntactic mulitlinear $(\Sigma\Pi)^\Delta\Sigma$ formula F of size at most s. By Lemma 14, we get that f can be expressed as a sum of at most $2s$ many summands, say $f_1, f_2, \ldots, f_s$ and $g_1, g_2, \ldots, g_s$, where each summand f_i is a t-product polynomial and each summand g_j is a t-simple polynomial for $t = \Omega(\Delta d^{1/\Delta})$.

For each $i \in [s]$, Lemma 16 implies that $\Pr[\mathrm{rank}\left(M_{(Y,Z)}(f_i|_\rho)\right) \geq 2^{m-\varepsilon t}] \leq \frac{1}{2^{\Omega(t)}}$ and $\Pr[\mathrm{rank}\left(M_{(Y,Z)}(g_i|_\rho)\right) \geq 2^{m-\delta t}] \leq \frac{1}{2^{\Omega(t)}}$, where ε and δ are absolute constants.

Thus, unless $s \geq 2^{\Omega(t)}$, we see by a union bound that there exists a ρ such that for each $i \in [s]$, $\mathrm{rank}\left(M_{(Y,Z)}(f_i|_\rho)\right) \leq 2^{m-\varepsilon t}$ and $\mathrm{rank}\left(M_{(Y,Z)}(g_i|_\rho)\right) \leq 2^{m-\delta t}$. For such a ρ, we have $\mathrm{rank}(M_{(Y,Z)}(F|_\rho)) \leq 2^m \cdot \left(\frac{s}{2^{\varepsilon t}} + \frac{s}{2^{\delta t}}\right) < 2^m$ unless $s \geq 2^{\Omega(t)}$.

From Lemma 16, we also know that for *any* choice of ρ in the sampling algorithm $\mathcal{S}$, we have $\mathrm{rank}(M_{(Y,Z)}(\mathrm{IMM}_d|_\rho)) \geq 2^m$. In particular, since F computes IMM_d, we must have $s \geq 2^{\Omega(t)} = 2^{\Omega(\Delta d^{1/\Delta})}$. ◀

Proof of Lemma 16.

Part 1: IMM_d has high rank. Let $\pi \in \{1,2\}^d$ and $a \in \{0,1\}^d$ be arbitrary. Note that in our sampling algorithm, ρ, A, b are completely determined given π and a.

Let us now examine the effect of ρ on IMM_d. We take the graph theoretic view of the polynomial IMM_d as given in Section 2.1.

Figure 2 illustrates how this restriction affects the variables labelling the edges of the graph G_d defined in Section 2.1. By substituting according to ρ in (1), we get that $\mathrm{IMM}_d(X)|_\rho = \prod_{i=1}^{m}(1+y_iz_i)$ if $|A| = 2m$ and $\prod_{i=1}^{m}(1+y_iz_i)\cdot(1+y_{m+1})$ if $|A| = 2m+1$, where $m = |Z|$. For any $S \subseteq [m]$, let Z_S (resp., Y_S) denote the monomial $\prod_{i\in S} z_i$ (resp., $\prod_{i\in S} y_i$). Now consider the matrix $M_{(Y,Z)}(\mathrm{IMM}_d|_\rho)$. We will simply use $\mathcal{M}$ to denote this matrix. For the sake of simplicity let us assume that $|A| = 2m$. (The case when $|A| = 2m+1$ is similar.) Let the rows and columns of $\mathcal{M}$ be labelled by the subsets of $[m]$ and let $\mathcal{M}(S,T)$ be the coefficient of $Y_S \cdot Z_T$ in $\mathrm{IMM}_d|_\rho$. It is easy to see that $\mathcal{M}(S,T) = 0$ if $S \neq T$ and 1 otherwise. That is, $\mathcal{M}$ is the Identity matrix of size $2^m \times 2^m$ and hence it has full rank.[7] ◁

Part 2: t-product polynomials have low rank. We now prove that for a t-product polynomial f, $\mathrm{rank}(M_{(Y,Z)}(f|_\rho))$ is small w.h.p.

Let f be a t-product polynomial, i.e. $f = f_1 f_2 \dots f_t$. Let $\chi : X \to [t]$ be a coloring function, which assigns colors to all the variables in X, so that $\chi^{-1}(i) = X_i^f$, where X_i^f is the variable set ascribed to f_i. That is, all the variables ascribed to f_i are assigned color i under the coloring function. To prove the lemma, we first show that, with high probability (over the choice of π), a constant fraction of the t colors appear along the path defined by π, i.e. along $(\pi(0),\pi(1)),(\pi(1),\pi(2)),\dots,(\pi(d-1),\pi(d))$. Given such a *multi-colored path*, we then show that with a high probability, over the choice of a, many of the colors have *an imbalance*. A color is said to have an imbalance under ρ if more variables from X of that color are mapped to the Y variables than the Z variables or vice versa. We then appeal to arguments similar to those in [15, 19, 5] to conclude that the imbalance results in a low rank.

Variable coloring, t-product polynomials and imbalance. We start with some notation. Given a string $\pi \in \{1,2\}^d$, let the path defined by π be the following sequence of pairs $(\pi(0),\pi(1))$, $(\pi(1),\pi(2)),\dots,(\pi(d-1),\pi(d))$ (we call it a path since these pairs correspond naturally to the edges of a path in the graph G_d defined in Section 2.1). We say that a color $\gamma \in [t]$ appears in layer $\ell \in [d]$ if there exist $u,v \in \{1,2\}$ such that $\gamma = \chi(x_{u,v}^{(\ell)})$.

Let $C^0 = \emptyset$ and let $C^i = C^{i-1} \cup \{\chi(x_{u,v}^{(i)}) \mid u,v \in \{1,2\}\}$ for $i \in [d]$, i.e., C^i contains all the distinct colors appearing in layers $\{1,2\dots,i\}$. Therefore, $|C^d| = t$. We will also define O^{2i+1} to be all the colors appearing in odd numbered layers up to $2i+1$, i.e. $O^{2i+1} = O^{2i-1} \cup \{\chi(x_{u,v}^{(2i+1)}) \mid u,v \in \{1,2\}\}$. Similarly, we define $E^{2i} = E^{2i-2} \cup \{\chi(x_{u,v}^{(2i)}) \mid u,v \in \{1,2\}\}$.

Let $C_\pi^0 = \emptyset$ and $C_\pi^i = C_\pi^{i-1} \cup \{\chi(x_{(\pi(i-1),\pi(i))}^{(i)})\}$, i.e. C_π^i contains all the distinct colors appearing along the path defined by π up to layer i. We first observe a property of C_π^d stated in the claim below.

▶ **Claim 17.** *If $|C^d| = t$, then $\Pr_\pi[|C_\pi^d| \le t/100] \le 1/2^{\Omega(t)}$.*

We will assume the claim and finish the proof of Part 2 of Lemma 16. The above claim shows that a lot of colors appear on the uniformly random path π with high probability. Using this, we will now show that a constant fraction of these colors also exhibit an imbalance with a high probability. Using the multiplicativity of the rank, we will then show that the imbalance for a large number of factors results in the low rank of the matrix $M_{Y,Z}(f|_\rho)$.

We will say that π is good if $|C_\pi^d| > t/100$. Let $L = t/100$. The above claim shows that a random π is good with high probability. In what follows, we condition on picking a good π. Let $a \in \{0,1\}^d$ be chosen uniformly at random as in the sampling algorithm. Let ρ be defined as in the sampling algorithm for π, a.

[7] If $|A| = 2m+1$ then $\mathcal{M}$ has a $2^m \times 2^m$ sized Identity matrix as a submatrix.

Let $\gamma \in C^d_\pi$ be a color that appears along π. Let π_γ be the elements along the path defined by π with color γ, i.e. $\pi_\gamma = \{(\pi(i-1),\pi(i)) \mid \chi(x^{(i)}_{(\pi(i-1),\pi(i))}) = \gamma\}$. Let $\rho(\pi_\gamma) = \{\rho(x^{(i)}_{(\pi(i-1),\pi(i))}) \mid (\pi(i-1),\pi(i)) \in \pi_\gamma\} \cap (Y \cup Z)$. A color $\gamma \in [t]$ is said to have an imbalance w.r.t. ρ if $||\rho(\pi_\gamma) \cap Y| - |\rho(\pi_\gamma) \cap Z|| \geq 1$.

It is easy to see that if $|\rho(\pi_\gamma)|$ is odd, then γ has an imbalance w.r.t. ρ. Note that the former event is equivalent to the event that $\bigoplus_{i \in P_\gamma} a_i$ equals 1 where $P_\gamma = \{i \mid (\pi(i-1),\pi(i)) \in \pi_\gamma\}$. Hence for any $\gamma \in C^d_\pi$, $\Pr[\gamma$ has an imbalance with respect to ρ along $\pi] = 1/2$. Further, as $|C^d_\pi| \geq L$ and the events corresponding to distinct $\gamma \in C^d_\pi$ are mutually independent, the Chernoff bound gives $\Pr[\leq L/4$ colors have an imbalance w.r.t. ρ along $\pi] \leq 1/2^{\Omega(L)}$. ◁

Assuming Claim 17 we are now done. The proof of Claim 17 can be found in [4].

Imbalance implies low rank. Let us recall that $f = f_1 f_2 \dots f_t$ is a t-product polynomial that is defined over the disjoint variable partition $X = X_1 \cup X_2 \cup \dots \cup X_t$ such that $|X_i| \geq 1$ for all $i \in [t]$. The following lemma (see, e.g., [19]) will be useful in bounding $\text{rank}(M_{(Y,Z)}(f|_\rho))$.

▶ **Lemma 18** ([19], Proposition 2.5). *Let $g = g_1 g_2 \cdots g_t$ be a t-product polynomial over the set of variables $Y \cup Z$ where $\text{Vars}(g_i) = Y_i \cup Z_i$. Then $\text{rank}(M_{(Y,Z)}(g)) = \prod_{i\in[t]} \text{rank}(M_{(Y_i,Z_i)}(g_i))$.*

From Lemma 18, we get that $\text{rank}(M_{(Y,Z)}(f|_\rho)) = \prod_{i=1}^t \text{rank}(M_{(Y_i,Z_i)}(f_i|_\rho))$ where $Y_i = Y \cap \{\rho(x) | x \in X_i\}$ and $Z_i = Z \cap \{\rho(x) | x \in X_i\}$. For all $i \in [t]$, from the definition it is clear that the rank of the matrix $M_{(Y_i,Z_i)}(f_i|_\rho)$ is upper bounded by $2^{\min\{|Y_i|,|Z_i|\}} \leq 2^{(|Y_i|+|Z_i|)/2}$. Let us note that these disjoint partitions in the t-product polynomial correspond to the colors in the coloring χ with all variables in X_i colored i. Hence if color i has imbalance w.r.t. ρ, then $\text{rank}(M_{(Y_i,Z_i)}(f_i|_\rho)) \leq 2^{\min\{|Y_i|,|Z_i|\}} \leq 2^{(|Y_i|+|Z_i|-1)/2}$. Thus, $\text{rank}(M_{(Y,Z)}(f|_\rho)) \leq \prod_{i=1}^t 2^{(|Y_i|+|Z_i|-1)/2} = 2^{((|Y|+|Z|)/2)-(\ell/2)} \leq 2^{m-(\ell-1)/2}$ where ℓ is the number of colors that have imbalance w.r.t. ρ. From the above discussion, we can infer that $\Pr_\pi[\text{rank}\,(M_{Y,Z}(f|_\rho)) \geq 2^{m-(t/1000)}] \leq \Pr_\pi[\ell \leq t/400] \leq \frac{1}{2^{\Omega(t)}}$.

Part 3: r-simple polynomials have low rank. Here we prove that if $f \in \mathbb{F}[X]$ is any r-simple polynomial, then for some absolute constant $\delta > 0$, $\Pr[\text{rank}(M_{(Y,Z)}(f|_\rho)) \geq 2^{m-\delta r}] \leq \frac{1}{2^{\Omega(r)}}$.

As f is an r-simple polynomial we know that $f = \left(\prod_{i=1}^{r'} L_i\right) \cdot G$, where $r' \leq r$, L_is are linear polynomials, $\forall i \in [r']$ X_i is the set of variables ascribed to L_i and $X_{r'+1}$ is the set of variables ascribed to G. Moreover, $|\cup_{i=1}^{r'} X_i| \geq 400r$.

To prove the above statement we set up some notation. Let $f|_\rho = \left(\prod_{i=1}^{r'} L_i|_\rho\right) \cdot G|_\rho$. Let $Y_i = \{\rho(x) \mid x \in X_i\} \cap Y$ and $Z_i = \{\rho(x) \mid x \in X_i\} \cap Z$ for each $i \in [r']$. Let $Y' = \cup_{i=1}^{r'} Y_i$ and $Z' = \cup_{i=1}^{r'} Z_i$. Also, let $Y'' = Y \setminus Y'$ and $Z'' = Z \setminus Z'$. Let U denote $\cup_{i=1}^{r'} X_i$ and let $U|_\rho = (\cup_{i=1}^{r'} Y_i) \cup (\cup_{i=1}^{r'} Z_i)$.

In the following claim we show that if U is a large set to begin with then with high probability (over the restriction ρ defined by the sampling algorithm), $U|_\rho$ is also large.

▶ **Claim 19.** *If $|U| \geq 400r$, then $\Pr[|U|_\rho| \leq 4r] \leq \frac{1}{2^{\Omega(r)}}$.*

We first finish the proof of Part 3 of Lemma 16 assuming this claim.

We say that a restriction ρ is good if we get $|U|_\rho| \geq 4r$. In what follows we will condition on the event that we have a good ρ. For a restriction ρ, for each $i \in [r']$, we can write $L_i|_\rho(Y_i, Z_i)$ as $L'_i|_\rho(Y_i) + L''_i|_\rho(Z_i)$ as L_is are linear polynomials. Therefore we get $\prod_{i=1}^{r'} L_i|_\rho(Y', Z') = \sum_{S \subseteq [r']} \prod_{i\in S} L'_i|_\rho(Y_i) \cdot \prod_{j\in[r']\setminus S} L''_j|_\rho(Z_j)$.

Let L_S denote the polynomial $\prod_{i\in S} L'_i|_\rho(Y_i)\cdot\prod_{j\in[r']\setminus S} L''_j|_\rho(Z_j)$. Note that for all $S\subseteq[r']$, $\operatorname{rank}\left(M_{(Y',Z')}(L_S)\right)$ is at most 1. Therefore, by the subadditivity of matrix rank, we get that $\operatorname{rank}\left(M_{(Y',Z')}\left(\prod_{i=1}^{r'} L_i|_\rho(Y',Z')\right)\right)\leq 2^{r'}\leq 2^r$. We can now bound $\operatorname{rank}\left(M_{(Y,Z)}\left(f|_\rho\right)\right)$.

$$\begin{aligned}\frac{\operatorname{rank}\left(M_{(Y,Z)}\left(f|_\rho\right)\right)}{2^{(|Y|+|Z|)/2}} &= \frac{\operatorname{rank}\left(M_{(Y,Z)}\left(\prod_{i=1}^{r'} L_i|_\rho\cdot G|_\rho\right)\right)}{2^{(|Y|+|Z|)/2}}\\ &= \frac{\operatorname{rank}\left(M_{(Y',Z')}\left(\prod_{i=1}^{r'} L_i|_\rho\right)\right)}{2^{(|Y'|+|Z'|)/2}}\cdot\frac{\operatorname{rank}\left(M_{(Y'',Z'')}\left(G|_\rho\right)\right)}{2^{(|Y''|+|Z''|)/2}}\\ &\leq \frac{2^r}{2^{|U|_\rho|/2}}\cdot 1\leq\frac{2^r}{2^{2r}}=\frac{1}{2^r}.\end{aligned}$$

where the second equality follows from Lemma 18. Therefore, we have $\operatorname{rank}\left(M_{(Y,Z)}\left(f|_\rho\right)\right)\leq 2^{(|Y|+|Z|)/2}/2^r\leq 2^{m+(1/2)-r}$ for any good ρ.

As Claim 19 tells us that ρ is good with probability $1-1/2^{\Omega(r)}$, we are done. ◁

Assuming Claim 19 (proof in [4]) we are done with the proof of Part 3 of Lemma 16. ◀

References

1. Manindra Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 67–75. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.32.
2. Eric Allender and Michal Koucký. Amplifying lower bounds by means of self-reducibility. *J. ACM*, 57(3):14:1–14:36, 2010. doi:10.1145/1706591.1706594.
3. Richard P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, 1974. doi:10.1145/321812.321815.
4. Suryajith Chillara, Nutan Limaye, and Srikanth Srinivasan. Small-depth multilinear formula lower bounds for iterated matrix multiplication, with applications. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:156, 2017. URL: https://eccc.weizmann.ac.il/report/2017/156.
5. Zeev Dvir, Guillaume Malod, Sylvain Perifel, and Amir Yehudayoff. Separating multilinear branching programs and formulas. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 615–624. ACM, 2012. doi:10.1145/2213977.2214034.
6. Hervé Fournier, Nutan Limaye, Guillaume Malod, and Srikanth Srinivasan. Lower bounds for depth 4 formulas computing iterated matrix multiplication. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 128–135. ACM, 2014. doi:10.1145/2591796.2591824.
7. Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Arithmetic circuits: A chasm at depth 3. *SIAM J. Comput.*, 45(3):1064–1079, 2016. doi:10.1137/140957123.
8. Pavel Hrubeš and Amir Yehudayoff. Homogeneous formulas and symmetric polynomials. *Computational Complexity*, 20(3):559–578, 2011.
9. Neeraj Kayal, Nutan Limaye, Chandan Saha, and Srikanth Srinivasan. An exponential lower bound for homogeneous depth four arithmetic formulas. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 61–70. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.15.

10 Neeraj Kayal, Vineet Nair, and Chandan Saha. Separation between read-once oblivious algebraic branching programs (roabps) and multilinear depth three circuits. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 46:1–46:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.46.

11 Neeraj Kayal, Chandan Saha, and Sébastien Tavenas. On the size of homogeneous and of depth four formulas with low individual degree. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 626–632. ACM, 2016. doi:10.1145/2897518.2897550.

12 Mrinal Kumar and Shubhangi Saraf. On the power of homogeneous depth 4 arithmetic circuits. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 364–373. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.46.

13 Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1997. doi:10.1007/BF01294256.

14 Ran Raz. Multilinear-nc neq multilinear-nc. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 344–351. IEEE Computer Society, 2004. doi:10.1109/FOCS.2004.42.

15 Ran Raz. Separation of multilinear circuit and formula size. *Theory of Computing*, 2(6):121–135, 2006. doi:10.4086/toc.2006.v002a006.

16 Ran Raz. Tensor-rank and lower bounds for arithmetic formulas. *J. ACM*, 60(6):40:1–40:15, 2013. doi:10.1145/2535928.

17 Ran Raz, Amir Shpilka, and Amir Yehudayoff. A lower bound for the size of syntactically multilinear arithmetic circuits. *SIAM J. Comput.*, 38(4):1624–1647, 2008. doi:10.1137/070707932.

18 Ran Raz and Amir Yehudayoff. Balancing syntactically multilinear arithmetic circuits. *Computational Complexity*, 17(4):515–535, 2008. doi:10.1007/s00037-008-0254-0.

19 Ran Raz and Amir Yehudayoff. Lower bounds and separations for constant depth multilinear circuits. *Computational Complexity*, 18(2):171–207, 2009. doi:10.1007/s00037-009-0270-8.

20 Benjamin Rossman. The average sensitivity of bounded-depth formulas. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 424–430. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.33.

21 Benjamin Rossman and Srikanth Srinivasan. Separation of acˆ0[oplus] formulas and circuits. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 50:1–50:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.50.

22 Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. Github survey, 2015. URL: https://github.com/dasarpmar/lowerbounds-survey/releases/.

23 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010. doi:10.1561/0400000039.

24 Philip M Spira. Computation times of arithmetic and boolean functions in (d, r) circuits. *IEEE Transactions on Computers*, 100(6):552–555, 1973.

25 Sébastien Tavenas. Improved bounds for reduction to depth 4 and depth 3. *Inf. Comput.*, 240:2–11, 2015. doi:10.1016/j.ic.2014.09.004.

26 Leslie G. Valiant, Sven Skyum, S. Berkowitz, and Charles Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.*, 12(4):641–644, 1983. doi:10.1137/0212043.

Upper and Lower Bounds for Dynamic Data Structures on Strings

Raphael Clifford
University of Bristol, Department of Computer Science, Bristol, U.K.
Raphael.Clifford@bristol.ac.uk

Allan Grønlund
Aarhus University, Department of Computer Science, Aarhus, Denmark
jallan@cs.au.dk

Kasper Green Larsen
Aarhus University, Department of Computer Science, Aarhus, Denmark
larsen@cs.au.dk

Tatiana Starikovskaya
École Normale Supérieure, Department of Computer Science, Paris, France
tat.starikovskaya@gmail.com

Abstract

We consider a range of simply stated dynamic data structure problems on strings. An update changes one symbol in the input and a query asks us to compute some function of the pattern of length m and a substring of a longer text. We give both conditional and unconditional lower bounds for variants of exact matching with wildcards, inner product, and Hamming distance computation via a sequence of reductions. As an example, we show that there does not exist an $O(m^{1/2-\varepsilon})$ time algorithm for a large range of these problems unless the online Boolean matrix-vector multiplication conjecture is false. We also provide nearly matching upper bounds for most of the problems we consider.

2012 ACM Subject Classification Theory of computation → Pattern matching, Theory of computation → Cell probe models and lower bounds, Theory of computation → Sketching and sampling

Keywords and phrases Exact Pattern Matching with Wildcards, Hamming Distance, Inner Product, Conditional Lower Bounds.

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.22

Funding Allan Grønlund and Kasper Green Larsen are supported by Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation, grant DNRF84.

1 Introduction

The search for lower bounds provides one of the greatest challenges in computer science. Progress in finding better truly unconditional lower bounds continues in slow but steady steps. There appears however, in the short term at least, to be no realistic prospect of finding unconditional lower bounds which are polynomial in the size of the input. One of the most exciting discoveries in recent years has been that such polynomial lower bounds can be given for a range of problems in **P** conditional on the hardness of a small set of well known and conjectured to be hard problems [2, 3, 13, 10, 1, 14]. These include the Strong Exponential Time Hypothesis (SETH), 3-SUM and online Boolean matrix-vector product (OMv).

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 22; pp. 22:1–22:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper we study the hardness of a number of simply stated dynamic string problems and show both conditional lower bounds based on the OMv conjecture (see Conjecture 14 for a precise statement) as well as unconditional lower bounds. We will also give new upper bounds which in many cases will nearly match our new conditional lower bounds. Each problem will have the following form.

▶ **Problem 1.** *Consider a text T of length n and a pattern P of length m. An update to the pattern (or text) is a pair (j, σ) which indicates that the letter at index j in the pattern (or text) is to be substituted with the letter σ. The task is to develop a dynamic data structure on P and T that maintains the following queries: Given a position i of T, output $f(P, T[i, \ldots, i + m - 1])$.*

Unless stated otherwise, we allow updates to both the pattern P and the text T. The different functions f we will consider are Hamming distance (DynHD), inner product (DynIP) and exact matching with wildcards (DynEM). These functions have formed the core of pattern matching with errors and wildcards for many years and have been extensively studied in both the standard offline pattern matching setting and to a lesser extent online and streaming. To the best of our knowledge, this is the first exploration of the complexity of pattern matching with errors and wildcards as a fully dynamic data structure problem.

By way of preparation, in Lemmas 4 and 5 we give $O(\sqrt{m \log m})$ query and update times for exact inner product, exact matching with wildcards, and for dynamic Hamming distance over constant-sized alphabets, as well as $O(m^{3/4} \log^{1/4} m)$-time algorithm for dynamic Hamming distance over polynomial-size alphabets. These algorithms are derived via a lazy rebuilding scheme. We then show in Theorem 15 that there does not exist an $O(m^{1/2-\epsilon})$ time solution to any of these problems unless the online Boolean matrix-vector conjecture is false. The lower bound for dynamic exact matching with wildcards is particularly interesting as it is exponentially higher than the known $O(\log m)$ time complexity for dynamic exact matching without wildcards.

Our conditional lower bound also extends to $(1+\varepsilon)$-approximate DynIP, DynIP modulo 2 and remarkably, to DynHD modulo 2 with a ternary input alphabet. This latter result is in stark contrast to the complexity of DynHD modulo 2 with a binary input alphabet which we show in Lemma 13 can be solved in $O(\log m / \log \log m)$ query and update time.

We complement all these lower bounds with a set of unconditional lower bounds derived via reductions from different 2d-dynamic range counting problems. First we show that DynIP is at least hard as weighted 2d-range counting. As a result our lower bound of $\Omega((\log m / \log \log m)^2)$ for DynIP matches the highest unconditional lower bound known for any dynamic data structure problem. We then go on to show $\Omega((\log^{1/2} m / \log \log m)^3)$ unconditional lower bounds for DynHD over binary alphabets, DynIP modulo 2 over binary alphabets and DynHD modulo 2 over ternary alphabets. These lower bounds are derived from a recent breakthrough in the complexity of the unweighted version of 2d-range counting. To finish our unconditional lower bounds we then show $\Omega(\log m / \log \log m)$ unconditional lower bounds for DynHD modulo 2 over binary alphabets, DynEM and $(1+\varepsilon)$-approximate DynIP.

As our final set of dynamic problems, we move on to consider $(1 + \varepsilon)$-approximate DynHD for which we do not have matching conditional lower bounds, despite its superficial similarity to approximate DynIP. Unlike for approximate DynIP and exact DynHD, in Section 4 we show markedly different upper bounds for approximate DynHD depending on whether updates may occur in only the pattern and text or in both. For the former case we derive $O(\varepsilon^{-c} \operatorname{polylog} m)$ time algorithms via Johnson-Lindenstrauss sketching. The exact value of c depends on the size of the input alphabet and in fact for some update operations

Table 1 Update/query time bounds for DynEM, DynHD, and DynIP for a text T of length $m \leq n \leq 2m$ and a pattern P of length m. For the conditional lower bounds, $\delta > 0$ is an arbitrary constant. Bounds for $(1+\varepsilon)$-approximate DynHD are not shown (see Section 4 for details).

	Mode	Alphabet	Upper bounds	Cond. lower bounds	Uncond. lower bounds
DynEM	exact	polynom.	$O(\sqrt{m \log m})$	$\Omega(m^{1/2-\delta})$	$\Omega(\log m / \log\log m)$
DynIP	exact	polynom.	$O(\sqrt{m \log m})$	$\Omega(m^{1/2-\delta})$	$\Omega((\log m / \log\log m)^2)$
	mod 2	$\{0,1\}$	$O(\sqrt{m \log m})$	$\Omega(m^{1/2-\delta})$	$\Omega((\log^{1/2} m / \log\log m)^3)$
	approx.	polynom.	$O(\sqrt{m \log m})$	$\Omega(m^{1/2-\delta})$	$\Omega(\log m / \log\log m)$
DynHD	exact	constant	$O(\sqrt{m \log m})$	$\Omega(m^{1/2-\delta})$	$\Omega((\log^{1/2} m / \log\log m)^3)$
		polynom.	$O(m^{3/4} \log^{1/2} m)$	$\Omega(m^{1/2-\delta})$	$\Omega((\log^{1/2} m / \log\log m)^3)$
	mod 2	$\{0,1\}$	$O(\log m / \log\log m)$	—	$\Omega(\log m / \log\log m)$
		$\{0,1,2\}$	$O(\sqrt{m \log m})$	$\Omega(m^{1/2-\delta})$	$\Omega((\log^{1/2} m / \log\log m)^3)$

the running time dependency on $\log m$ is completely removed. For the latter case with updates in both the pattern and text, our upper bound is $O(\varepsilon^{-2}\sqrt{m}\ \text{polylog}\ m)$ time. It is an interesting and open question whether there exist matching conditional lower bounds for these versions of approximate DynHD as well. We give a summary of the results in Table 1.

2 Related work

In the dynamic setting we consider with single character updates, the most closely related previous work considers the problem of dynamic exact matching. In [8] an $O(\log\log m)$ time algorithm was shown for dynamic exact matching when updates are only permitted in the text [8]. In [5] a more general data structure was developed supporting insertion and deletion of characters and movements of arbitrary large blocks of text. This was improved in a succession of papers culminating in the work [19] who give a data structure that supports, amongst other properties, concatenation, splitting and equality testing in $O(\log m)$ update and $O(1)$ query time. The same data structure solves, for example, the dynamic exact matching problem without wildcards problem in $O(\log m)$ time. At the expense of $O(\log^2 m)$ updates this latter work also supports finding occurrences of a specified pattern P in $O(|P|)$ time. A separate line of work has considered the static data structure problem of text indexing for approximate matching [11, 7, 17, 12, 21, 12, 9].

3 Upper bounds for DynHD, DynIP, and DynEM

In this section we show upper bounds for DynIP, DynHD, and DynEM problems. Recall that a query i asks for $f(P, T[i, \ldots, i+m-1])$. For DynIP we define $f(P, T[i, \ldots, i+m-1])$ to be equal to the inner product of P and $T[i, \ldots, i+m-1]$, for DynHD the Hamming distance between P and $T[i, \ldots, i+m-1]$. In the DynEM problem we assume that P and T are strings over $\Sigma \cap \{?\}$, where Σ is an integer alphabet and ? is a special wildcard symbol that matches any letter in Σ. We define $f(P, T[i, \ldots, i+m-1])$ to be equal to zero if P matches $T[i, \ldots, i+m-1]$ and the number of mismatching positions otherwise. We define n to be the length of the text, and m to be the length of the pattern, $n \geq m$.

We will in fact present a general solution for dynamic string problems where f can be represented in a particular form. DynIP, DynHD and DynEM will seen as special cases.

The restriction is simply that $f(P, T[i, \ldots, i+m-1]) = \sum_{j=1}^{j=m} g(P[j], T[i+j-1])$, where the function g can be evaluated in constant time. This functional form is closely related to the idea of *local* distance functions that were key to the development of fast streaming pattern matching algorithms [16]. We first show that our string problems do indeed satisfy the stated requirements.

▶ **Lemma 2.** *If f is inner product, Hamming distance, or exact matching with wildcards, then there exists a function g such that $f(P, T[i, \ldots, i+m-1]) = \sum_{j=1}^{j=m} g(P[j], T[i+j-1])$, where the function g can be evaluated in constant time.*

Proof. If f is inner product, we put $g(P_j, T_{i+j-1}) = P_j \cdot T_{i+j-1}$. In the case of Hamming distance, we define $g(P_j, T_{i+j-1}) = 0$ if $P_j = T_{i+j-1}$ and $g_j(P_j, T_{i+j-1}) = 1$ otherwise.

For DynEM we assume that wildcards are represented by the value 0. It is not hard to see that we can take $g(P_j, T_{i+j-1})$ to be the characteristic function of $(P_j - T_{i+j-1})^2 P_j T_{i+j-1} > 0$ and indeed this observation is the basis for one of the fastest offline exact matching with wildcards algorithms [15]. The key property we use is that either (a) if one of P_j and T_{i+j-1} is a wildcard or $P_j = T_{i+j-1}$ then $g(P_j, T_{i+j-1}) = 0$, or (b) $P_j \neq T_{i+j-1}$ and then $g(P_j, T_{i+j-1}) > 0$. It follows that $f(P, T[i, \ldots, i+m-1])$ equals zero if and only if P and $T[i, \ldots, i+m-1]$ match. ◀

We now show a solution for all dynamic string problems defined by a function f that can be represented in the form above. We consider the most general update model, where we are allowed to update both the text and the pattern.

▶ **Theorem 3.** *Let T be a text of length n, and P be a pattern of length m. Assume f can be represented as $f(P, T[i, \ldots, i+m-1]) = \sum_{j=1}^{j=m} g(P_j, T_{i+j-1})$, where g can be computed in constant time, and the values $f(P, T[1, \ldots, m])$, $f(P, T[2, \ldots, m+1])$, $\ldots$, $f(P, T[n-m+1, \ldots, n])$ can be computed in $\mathcal{T}(n)$ time and $S(n)$ space. We can then solve the corresponding dynamic string problem in $O(\sqrt{\mathcal{T}(n)})$ worst case update/query time using $O(S(n)+n)$ space.*

Proof. Let us first show a solution with $O(\sqrt{\mathcal{T}(n)})$ amortised time. We start by computing values $A[1] = f(P, T[1, \ldots, m]), \ldots, A[n-m+1] = f(P, T[n-m+1, \ldots, n])$ in $O(\mathcal{T}(n))$ time and $S(n)$ space. At all times, we maintain a list of updates U that have occurred since the last moment we recomputed the values $A[i]$. Suppose that the size of U is at most $\lceil\sqrt{\mathcal{T}(n)}\rceil$ and a query i arrives. We can then compute $A'[i] = f(P, T[i, \ldots, i+m-1])$ from $A[i]$ and U in the following way. We initialise $A'[i] = A[i]$, and consider each update in order. Suppose that an update change letters in a position k of P or $T[i, \ldots, i+m-1]$, and let P'_k and T'_{i+k-1} be the updated letters. We remember P'_k and T'_{i+k-1}, and set

$$A'[i] \leftarrow A'[i] - g(P_k, T_{i+k-1}) + g(P'_k, T'_{i+k-1})$$

Since g can be evaluated in constant time, this step takes constant time as well. Therefore, the time to perform each query is $O(\sqrt{\mathcal{T}(n)})$. When the size of U reaches $\lceil\sqrt{\mathcal{T}(n)}\rceil$, we apply the updates in U to T and P, empty U, and recompute the values $A[i]$ from scratch. The amortised cost of an update is therefore $O(\sqrt{\mathcal{T}(n)})$.

We can de-amortise the solution in a standard way. Namely, we restart the computation of the values $A[i]$ each $\lceil\sqrt{\mathcal{T}(n)}/2\rceil$ updates, and run $\Theta(\sqrt{\mathcal{T}(n)})$ steps of the computation per each of the $\lceil\sqrt{\mathcal{T}(n)}/2\rceil$ subsequent updates. While the computation is not over, we use the previously computed values $f(P, T[1, \ldots, m]), \ldots, f(P, T[n-m+1, \ldots, n])$ to answer queries. As before, we will need to correct the value of the function g in at most

$\lceil\sqrt{\mathcal{T}(n)}/2\rceil$ positions. Note that apart from the space we need for computing the values $f(P, T[1,\ldots,m])$, $f(P, T[2,\ldots,m+1])$, $\ldots$, $f(P, T[n-m+1,\ldots,n])$, we need only $O(n)$ space. ◀

Let us first assume that $m \leq n \leq 2m$, later we will show how to extend the solution to a general value of n when the updates occur only in the text.

▶ **Lemma 4.** *For a text T of length $m \leq n \leq 2m$, and a pattern P of length m, problem* DynHD *can be solved in $O(\sqrt{m \log m})$ query/update time for constant-size alphabets, and in $O(m^{3/4} \log^{1/4} m)$ query/update time for polynomial-size alphabets. Both solutions use $O(m)$ space.*

Proof. If the alphabet is binary, the values $f(P, T[1,\ldots,m]),\ldots, f(P, T[n-m+1,\ldots,n])$ can be computed by running the FFT algorithm twice. Recall that the FFT algorithm computes the inner product for each alignment of two strings. By running the FFT algorithm on P and T for the first time, we obtain, for each i, the number of positions j such that $P[j] = T[i+j] = 1$. By running it for the second time on the copies P and T where each bit is flipped, we obtain, for each i, the number of positions j such that $P[j] = T[i+j] = 0$. We can then compute the values $f(P, T[1,\ldots,m]),\ldots, f(P, T[n-m+1,\ldots,n])$ in linear time. For this algorithm, $\mathcal{T}(n) = O(n \log n) = O(m \log m)$. For alphabets of constant size $|\Sigma|$, we run the FFT algorithm $|\Sigma|$ times, once for each letter $a \in \Sigma$, on the copies of P and T where a is replaced with 1 and all letters in $\Sigma \setminus \{a\}$ are replaced with 0. $T(n) = O(m \log m)$ as well. For polynomial-size alphabets, $\mathcal{T}(n) = O(n\sqrt{n \log n}) = O(m\sqrt{m \log m})$ and $S(n) = O(n) = O(m)$ bounds were shown independently by Abrahamson [4] and Kosaraju [24] in 1987. The claim immediately follows from Lemma 2 and Theorem 3. ◀

▶ **Lemma 5.** *For a text T of length $m \leq n \leq 2m$, and a pattern P of length m, problems* DynIP *and* DynEM *can be solved in $O(\sqrt{m \log m})$ query/update time using $O(m)$ space.*

Proof. For both problems, $\mathcal{T}(n) = O(n \log n) = O(m \log m)$ and $S(n) = O(n) = O(m)$. For inner product, this is a direct corollary of the FFT algorithm. The bound for exact matching with wildcards was demonstrated in [18, 15]. The claim follows from Lemma 2 and Theorem 3. ◀

We now extend our solution to a general value of n in the case where only updates to the text are allowed. In this case there is also an additional cost of computing the full set of solutions before the first query or update is performed which we omit from the following theorem.

▶ **Theorem 6.** *For a text T of length $n \geq m$, and a pattern P of length m, problem* DynHD *can be solved in $O(\sqrt{m \log m})$ query/update time for constant-size alphabets, and in $O(m^{3/4} \log^{1/4} m)$ query/update time for polynomial-size alphabets. Both solutions use $O(n)$ space. Problems* DynIP *and* DynEM *can be solved in $O(\sqrt{m \log m})$ query/update time using $O(n)$ space.*

Proof. We first partition T into blocks of length $2m$ overlapping by m positions (the last block may be shorter). Note that for each i a string $T[i,\ldots,i+m-1]$ is a substring of one of such blocks, and each position of T belongs to at most two blocks. It follows that if we have a solution for an m-length block with update time t_u, query time t_q, and space S, then we have a solution for T with update time $O(t_u)$, query time t_q, and space $O(\frac{n}{m} \cdot S)$. The claim follows from Lemmas 4 and 5. ◀

4 Upper bounds for dynamic approximate Hamming distance

In this section we develop algorithms for an approximate version of DynHD, which we refer to as DynApproxHD. In this problem a query i must return a $(1+\varepsilon)$-approximation of the Hamming distance between P and $T[i,\ldots,i+m-1]$, where $\varepsilon > 0$ is a parameter of the algorithm. Unlike the other problems we have considered, the complexity of DynApproxHD appears to have a strong dependence on whether updates are permitted only in the pattern or text or in both. At one extreme, when updates are only permitted in the pattern and the input alphabet is binary, we show in Theorem 9 a data structure that takes $O(1/\varepsilon)$ update and $O(1/\varepsilon^2)$ query time. However if updates can occur in both the pattern and the text, then the complexity increases dramatically to be at least that of exact DynIP, DynEM and DynHD over binary alphabets.

In Section 3 we showed that the DynHD problem can be solved in $O(m^{1/2}\log^{1/2} m)$ query/update time for constant-size alphabets, and in $O(m^{3/4}\log^{1/4} m)$ query/update time for polynomial-size alphabets. We start our exploration of the complexity of DynApproxHD by showing that this dependence on the alphabet size is almost completely removed in this approximate setting. The solution we give is deterministic and is based on the mapping idea of Karloff [23].

▶ **Lemma 7** ([23]). *Let Σ be the alphabet of P and T. There exists $\Theta((1/\varepsilon^2)\log^2 n)$ deterministic mappings $map_j : \Sigma \to \{0,1\}$ such that a $(1+\varepsilon)$-approximation of the Hamming distance between P and T at a particular alignment can be given by a normalised average of the Hamming distances between $map_j(P) = map_j(P_1)\ldots map_j(P_n)$ and $map_j(T) = map_j(T_1)\ldots map_j(T_n)$ at this alignment. Each mapping can be stored as a look-up table that permits to compute each $map_j(P_k)$ or $map_j(T_k)$ in $O(1)$ time.*

▶ **Corollary 8.** *For a text T of length $m \le n \le 2m$, and a pattern P of length m, the* DynApproxHD *problem over polynomial-size alphabets can be solved in $O((1/\varepsilon^2)\sqrt{m}\cdot\text{polylog } m)$ query/update time and $O((1/\varepsilon^2)m\log^2 m)$ space.*

Proof. We consider Karloff's mappings map_j. For each j, we run our DynHD solution for constant-size alphabets (Lemma 4) on $map_j(P)$ and $map_j(T)$. The claim immediately follows. ◀

We now present several randomised solutions for DynApproxHD in two special update models where we are allowed to update either only the text or only the pattern. We first assume a binary input alphabet, and then show how to extend our solutions to constant-size and then later polynomial-size alphabets as well.

▶ **Theorem 9.** *For a text T of length $n \ge m$, and a pattern P of length m, there is a randomised data structure for the* DynApproxHD *problem over a constant-sized alphabet with*

(a) *$O(1/\varepsilon)$ update time, $O(1/\varepsilon^2)$ query time, and $O((1/\varepsilon^2)\cdot n)$ space if only updates to the pattern are allowed;*

(b) *$O((1/\varepsilon)\cdot\text{polylog } n)$ update time and $O((1/\varepsilon^2)\cdot\text{polylog } n)$ query time using $O((1/\varepsilon^2)\cdot n\ \text{polylog}\ n)$ space if only updates to the text are allowed.*

Each answer is correct with constant probability.

Proof. Let us first assume the input alphabet is of constant size. We will make use of the sparse Johnson-Lindenstrauss transform by Kane and Nelson [22] defined by a random $\Theta(1/\varepsilon^2)\times n$ matrix M such that its entries are from $\{-1,0,1\}$, and each of its columns

contains $\Theta(1/\varepsilon)$ non-zero entries. The result of a transform, which we call a sketch, is defined to be equal to $M \cdot x$. Kane and Nelson showed how to choose a distribution on such matrices such that, with constant probability, the appropriately scaled square of the L_2 norm of the difference of the sketches of two strings gives a $(1+\varepsilon)$-approximation of Hamming distance.

(a) During the preprocessing step we compute the sketch of P and of each m-length substring of T. When an update to P arrives, we update its sketch in a naive way in $O(1/\varepsilon)$ time. When a query i arrives, we can compute a $(1+\varepsilon)$-approximation of the Hamming distance between P and T by computing the L_2 norm of the difference of the sketches of P and $T[i,\ldots,i+m-1]$. Since the sketches are the vectors of length $1/\varepsilon^2$, this can be done in $O(1/\varepsilon^2)$ time.

(b) For this model, we will need a sketch that gives $(1+\varepsilon)$-approximation of Hamming distance with error probability $\Theta(1/\log m)$. This can be achieved by repeating the scheme $\Theta(\log\log m)$ times. During the preprocessing, we first compute $\Theta(\log\log m)$ sketches for each 2^k-length substring of the pattern P, where $k = 1, 2, \ldots, \log m$. We then compute $\Theta(\log\log m)$ sketches for each substring $T[i\cdot 2^k+1,\ldots,(i+1)\cdot 2^k]$. We call such substrings of T canonical. When an update (i,σ) arrives, we need to fix the sketches of $O(\log m)$ canonical substrings (since T_i belongs to $O(\log m)$ such substrings), which can be done in $O((1/\varepsilon)\log m\log\log m)$ time. A query i can be answered in $O((1/\varepsilon^2)\log m\log\log m)$ time: First, we partition $T[i,\ldots,i+m-1]$ into $O(\log m)$ canonical substrings $S_1,\ldots,S_k$. Secondly, we compute a $(1+\varepsilon)$-approximation of the Hamming distance between each S_i and the corresponding substring of P using the sketches. Finally, we sum up all approximations to obtain the answer. Since the probability to error on each pair of substrings is $\Theta(1/\log m)$, the total error probability is constant by the union bound.

Both algorithms can be extended to work for any constant sized alphabet by expanding the input alphabet in unary. That is we replace the letter i with a binary vector $0\ldots010\ldots0$, where the set bit is in the i-th position. ◀

▶ **Corollary 10.** *For a text T of length $n \geq m$, and a pattern P of length m, and $\varepsilon > 1/n$, there is a randomised data structure for the* DynApproxHD *problem over polynomial-size alphabets with*

(a) *$O((1/\varepsilon^3)\cdot\text{polylog}\, n)$ update time, $O((1/\varepsilon^4)\cdot\text{polylog}\, n)$ query time, and $O((1/\varepsilon^4)\cdot n\,\text{polylog}\, n)$ space if only updates to the pattern are allowed;*

(b) *$O((1/\varepsilon^4)\cdot\text{polylog}\, n)$ update time, $O((1/\varepsilon^4)\cdot\text{polylog}\, n)$ query time, and $O((1/\varepsilon^4)\cdot n\,\text{polylog}\, n)$ space if only updates to the text are allowed.*

Each answer is correct with constant probability.

Proof. We reduce the alphabet to binary by applying Karloff's mappings. There are $\Theta((1/\varepsilon^2)\log^2 n)$ mappings, and to compute the Hamming distance between P and $T[i,\ldots,i+m-1]$ we need to compute the Hamming distance for each pair $map_j(P)$ and $map_j(T[i,\ldots,i+m-1])$. To achieve constant error probability, we run $\Theta(\log((1/\varepsilon)\log n)) = \text{polylog}\, n$ instances of the algorithm for text-only or pattern-only updates (Theorem 9). (We note that we will achieve $(1+\varepsilon)^2$-approximation, which is $(1+\varepsilon')$-approximation for $\varepsilon' = 2\varepsilon+\varepsilon^2$.) ◀

5 Lower bounds

In this section we demonstrate conditional and unconditional lower bounds for different variants of DynEM, DynIP, and DynHD. The conditional lower bounds are derived from

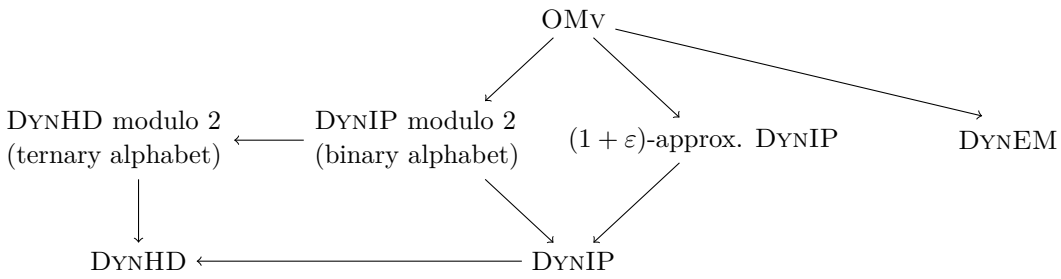


Figure 1 Reductions between OMv and different variants of DynEM, DynIP, and DynHD.

the hardness of a well-known problem, online Boolean matrix-vector product (OMv). Fig. 1 summarises the reductions we use.

5.1 Reductions between DynIP, DynHD and DynHD modulo 2

Before we get to our main lower bounds results we will first establish the relationship between some of the dynamic string problems we consider.

▶ **Lemma 11.** DynHD *is at least as hard as* DynIP *over binary alphabets.*

Proof. We map the input alphabet of the text and the pattern separately. Take an instance of DynIP where the input alphabet is binary. In order to transform it into an instance of DynHD each 1 in the pattern or text is mapped to the string 111 in the DynHD instance. Similarly, a 0 in the pattern is mapped to the string 010 and a 0 in the text is mapped to the string 100. This transformation ensures that any two symbols that align in the DynIP instance will give Hamming distance 2 in the DynHD instance except when two 1s align. In this case the Hamming distance will be 0. We can therefore infer the inner product from the Hamming distance: The inner product will be equal to the length of the pattern minus the Hamming distance divided by two. ◀

We will later show both conditional and unconditional lower bounds not only for DynIP but also for DynIP modulo 2. The following two lemmas will lead to perhaps our most surprising result which is that DynHD modulo 2 over *ternary* alphabets is exponentially harder to solve than DynHD modulo 2 over a binary alphabet. It is worth emphasising by way of contrast that in the standard offline pattern matching setting, the asymptotic complexity of computing the Hamming distance at all alignments of a pattern and text is identical for any constant sized input alphabet.

▶ **Lemma 12.** DynHD *modulo* 2 *over a ternary alphabet is at least as hard as* DynIP *modulo* 2 *over a binary alphabet.*

Proof. We again map the input alphabet of the text and pattern separately. Take an instance of DynIP modulo 2 where the input alphabet is binary. Each 1 in the pattern is mapped to the string 22 and each 0 in the pattern is mapped to the string 01. Each 1 in the text is mapped to the string 11 and each 0 in the text is mapped to the string 02. This transformation ensures that any two symbols that align in the DynIP modulo 2 instance will give Hamming distance 1 in the DynHD modulo 2 instance except for when two 1s align in the DynIP modulo 2 instance when the resulting Hamming distance is 2. Therefore, the inner product modulo 2 is equal to the length of the pattern minus the Hamming distance modulo 2. ◀

However, DynHD modulo 2 over a binary alphabet is much easier than DynHD modulo 2 over a ternary alphabet.

▶ **Lemma 13.** *For a binary text T of length $n \geq m$, and a binary pattern P of length m the* DynHD *modulo* 2 *problem can be solved in $O(\log m / \log\log m)$ update/query time using $O(n)$ space. There is a matching unconditional lower bound for update/query time as well.*

Proof. As before, we divide the text T into $2m$-length blocks overlapping by m positions. We will show that for each block DynHD modulo 2 can be solved in $O(\log m / \log\log m)$ update/query time using $O(m)$ space, hence giving the claim.

Consider a $2m$-length block of T. In order to answer a query at alignment i for DynHD modulo 2 we need only to sum, modulo 2, the number of 1s in the pattern and the corresponding substring of the text $T[i, \ldots, i+m-1]$. This can be seen via a simple proof by induction as follows. As the base case consider two strings of length 1 and let all arithmetic be over $\mathbb{Z}_2$. In this case the Hamming distance is the sum of the Hamming weights of the two strings. For the inductive step, extend each of these two strings by one bit and observe that the new Hamming distance is the old Hamming distance before extending the strings plus the sum of the two new bits over $\mathbb{Z}_2$.

The Hamming weight of the pattern can be maintained straightforwardly. We argue that answering queries for the Hamming weight of substrings of the block is equivalent to the prefix sum problem modulo 2. To reduce from this problem to prefix sum we need only observe that we can compute the number of 1s in $T[i, \ldots, i+m-1]$ by subtracting the prefix sum up to index $i-1$ from the prefix sum up to index $i+m-1$. To reduce from prefix sum to the DynHD modulo 2 problem we construct a text of length $2m$ with the first half all zeros and the second half as a copy of the prefix sum array. Setting the pattern to all 1s we can compute the prefix sum modulo 2 up to index i of its array of length m by performing a query at index i of the text. It follows from the upper and lower bounds of [28] that the complexity of DynHD modulo 2 over a binary alphabet is $\Theta(\log m / \log\log m)$. ◀

5.2 Conditional lower bounds

We will now give lower bounds for our dynamic string problems conditional on the hardness of a well known problem. The OMv problem was introduced in [20] as a means to prove conditional lower bounds for a number of dynamic problems. In this problem we are first given an $r \times r$ Boolean matrix M. We then receive r vectors $v_1, \ldots, v_r$, one by one. After seeing each vector v_i, we have to output the product Mv_i (over the Boolean semi-ring) before we receive the next vector. A naive algorithm can solve this problem using $O(r^3)$ time in total with the current fastest solution taking $O(r^3/2^{\Omega(\sqrt{\log r})})$ time [27]. The OMv conjecture is as follows:

▶ **Conjecture 14** (OMv Conjecture [20])**.** *For any constant $\epsilon > 0$, there is no $O(r^{3-\epsilon})$-time algorithm that solves the* OMv *problem with error probability of at most* 1/3*.*

▶ **Theorem 15.** *Assuming the* OMv *conjecture, there does not exist an algorithm running in $O(m^{1/2-\epsilon})$ for the maximum of query and update time for* DynEM*,* DynIP*, and* DynHD*. The same lower bound holds for* DynIP *modulo* 2*, for $(1+\varepsilon)$-approximate* DynIP*, and for* DynHD *modulo* 2 *over ternary alphabets. The same lower bound holds even when updates are permitted only in the pattern or only in the text.*

Proof. We first give a reduction from the online Boolean matrix-vector multiplication problem to DynEM. We create a text T of length $2m = 2r^2$ from the matrix M by concatenating

the r rows of M one after another and filling the rest of T with the symbol 1 repeated r^2 times. Now consider a single Boolean matrix vector product Mv_i. The pattern P has length $m = r^2$. Its first r symbols are a copy of the vector v_i but with all 0s replaced by the wildcard symbol ? and all 1s replaced by the symbol 0. The remaining $r^2 - r$ symbols are set to the wildcard symbol ?. To perform a Boolean matrix vector multiplication we perform m exact match with wildcard queries at indices $1, r+1, 2r+1, \ldots, (r-1)r+1$. If a query i returns a match then $Mv_i[j] = 0$ and $Mv_i[j] = 1$ otherwise. If follows that any algorithm for DynEM running in $O(m^{1/2-\varepsilon})$ for the maximum of query and update time implies an $O(r^{3-\varepsilon})$-time algorithm that solves the online Boolean matrix-vector multiplication problem, thereby contradicting the OMv conjecture.

DynIP and DynHD are at least as hard as DynIP modulo 2, so it suffices to show the lower bound for the latter. We give a similar reduction from OMv but this time with an extra randomisation step. We create a text T of length $2m = 2r^2$ from the matrix M by concatenating the r rows of M one after another and filling the rest of T with the symbol 0 repeated r^2 times. Now consider a single Boolean matrix vector product Mv_i. We create a pattern P of length $m = r^2$ with the first r symbols being a copy of v_i and the remaining $r^2 - r$ symbols set to 0. We now flip each set bit in P with probability $1/2$ and compute inner product modulo 2 queries at indices $1, r+1, 2r+1, \ldots, (r-1)r+1$. If $Mv_i[j] = 0$ then an inner product query j will always return 0. If $Mv_i[j] = 1$ then the inner product query will return 1 with probability $1/2$. This gives a probability of at least $1/2$ of giving the correct answer for each $Mv_i[j]$. We amplify the probabilities by repeating the randomised procedure $O(\log m)$ times using the fact that we have one-sided error at each iteration. It then follows that there does not exist an algorithm running in $O(m^{1/2-\varepsilon})$ for the maximum of query and update time for DynIP modulo 2 unless the OMv conjecture is false.

The lower bound for $(1+\varepsilon)$-approximate DynIP follows from the same reduction with the arithmetic performed over the reals rather than modulo 2 and without the randomisation step. This is because a $(1+\varepsilon)$-approximation must be able to distinguish zero and non-zero inner products which is sufficient for our reduction from OMv.

The lower bound for DynHD modulo 2 over a ternary alphabet now follows from Lemma 12.

If updates are only allowed in the text then we derive the same lower bound as before by modifying our reductions. Let us take the reduction from the online Boolean matrix-vector multiplication problem to DynIP modulo 2 as an example. The other lower bounds follow analogously. We create a pattern P of length $m = r^2$ from the matrix M by concatenating the r rows of M one after another. The text is of length $2m = 2r^2$ and will be all 0s except for the substring $T[r^2 - r + 1, \ldots, r^2]$. In order to perform a single Boolean matrix vector product Mv_i the substring is updated so that $T[r^2 - r + 1, \ldots, r^2] = v_i$ and we then flip each set bit in T with probability $1/2$. We then compute inner product queries modulo 2 at indices $1, r+1, 2r+1, \ldots, (r-1)r+1$ which give the correct answer for each query with probability at least $1/2$. We can amplify the probability as before giving us the desired lower bound. ◀

Our lower bound also holds for DynIP modulo c for any $c \geq 2$.

▶ **Corollary 16.** *Let integer $c \geq 2$. Assuming the* OMv *conjecture, there does not exist an algorithm running in $O(m^{1/2-\epsilon})$ for the maximum of query and update time for* DynIP *modulo c.*

Proof. Let the input alphabet be binary as before and perform the same randomised reduction from OMv as in the proof of Theorem 15. If the inner product equals 0 then we always

give the correct answer. If the inner product is greater than 0 then after flipping the set bits, the inner product modulo c is greater than 0 with probability that tends asymptotically to $\frac{c-1}{c}$. We can then amplify the probabilities to ensure that every value in the matrix-vector product is correct with constant probability as before. ◀

5.3 Unconditional lower bounds

In this section we will give unconditional lower bounds for all the problems we have considered except DynApproxHD. Although these bounds are necessarily much lower than the conditional lower bounds we gave previously, they nonetheless match in many cases the limits of what is known unconditionally for any dynamic data structure.

We first show lower bounds for the DynIP and the DynHD problems by reduction from the dynamic weighted range counting problem. In this problem, we are given a $r \times r$ grid D. The points in the grid are assigned integer weights, and at any moment there can be at most r non-zero weights w_i. For our problem $r = m^{1/3}$. Updates may change the weight of a point and a query (i, j) asks for $\sum_{x \leq i, y \leq j} D_{x,y}$. In [25] Larsen gave an $\Omega((\log r / \log\log r)^2)$ lower bound for the maximum of query and update time for dynamic weighted range counting. This lower bound does not hold however in the unweighted case (where the weights are in $\{0, 1\}$) and giving an $\omega(\log r)$ lower bound for this situation remained an important open problem for a number of years. Recently in [26] a new $\Omega((\log^{1/2} r / \log\log r)^3)$ lower bound was given for this unweighted range counting problem which also holds over $\mathbb{F}_2$.

▶ **Theorem 17.** *The* DynIP *problem has an unconditional* $\Omega((\log m / \log\log m)^2)$ *lower bound for the maximum of query and update time for polynomial-size alphabets.* DynHD *over binary alphabets,* DynIP *modulo* 2 *over binary alphabets and* DynHD *modulo* 2 *over ternary alphabets have an* $\Omega((\log^{1/2} m / \log\log m)^3)$ *lower bound.*

Proof. We give a reduction from dynamic range counting to DynIP. We take an instance of the problem for $r = m^{1/3}$ and create a text T of length $2m$ and a pattern P of length m. The text has all symbols set to 0 except $T_{m-m^{1/3}+1}, \ldots, T_m$ that are set to $w_1, \ldots, w_{m^{1/3}}$ respectively. For each of the $m^{2/3}$ different possible queries to D, a subset of the w_i's will be included in the query. We create a pattern P so that $P_{jm^{1/3}+i-1} = 1$ if weight w_i is included in the range for query j and $P_{jm^{1/3}+i-1} = 0$ otherwise.

To perform a range counting query, we need to align the relevant substring of the pattern of length $m^{1/3}$ with $T[m - m^{1/3} + 1, \ldots, m]$ and perform an inner product query. Our lower bounds then follow from the lower bounds for the weighted and $\mathbb{F}_2$ versions of dynamic range counting and Lemmas 11 and 12. ◀

Finally, we give lower bounds for the DynEM and the $(1 + \varepsilon)$-approximate DynIP problems by reduction from the dynamic range emptiness problem. In this problem, the set-up is exactly like in the unweighted dynamic range counting problem above, and a query (i, j) asks if $\sum_{x \leq i, y \leq j} D_{x,y} = 0$. In [6], Alstrup et al. showed a $\Omega(\log r / \log\log r)$ lower bound for this problem.

▶ **Theorem 18.** *Both the* DynEM *and the* $(1 + \varepsilon)$*-approximate* DynIP *problems have unconditional* $\Omega(\log m / \log\log m)$ *lower bounds for the maximum of query and update time.*

Proof. Consider an instance of two dimensional range emptiness on D for $r = m^{1/3}$. We take an instance of this problem and create a text T of length $2m$ and a pattern P of length m. The text has all values set to 0 except $T_{m-m^{1/3}+1}, \ldots, T_m$ set to $w_1, \ldots, w_{m^{1/3}}$ respectively. For each of the $m^{2/3}$ different possible queries to D in the dynamic range

emptiness problem, a subset of the w_i's will be included in the query. We create a pattern P so that $P_{jn^{1/3}+i-1} = 0$ if weight w_i is included in the range for query j and $P_{jn^{1/3}+i-1} = ?$ otherwise. If an exact match with wildcards query returns True then we know that all the weights in the corresponding range are 0. If it returns False then we know the range is not empty. We therefore have reduced from two dimensional range emptiness to DynEM giving an $\Omega(\log m/\log\log m)$ lower bound for DynEM.

For the $(1+\varepsilon)$-approximate dynamic inner product problem we must be able to distinguish an inner product of zero from all other values. We therefore use the same reduction from the proof of Theorem 17 but this time only report whether the approximate inner product is greater than zero. The result of this query is sufficient to determine the answer to a range emptiness query and we therefore derive the same $\Omega(\log m/\log\log m)$ lower bound. ◀

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.14.

2 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.

3 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2014. doi:10.1007/978-3-662-43948-7_4.

4 Karl R. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987. doi:10.1137/0216067.

5 Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Pattern matching in dynamic texts. In *SODA '00: Proc. 11th ACM-SIAM Symp. on Discrete Algorithms*, pages 819–828, 2000.

6 Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *FOCS '98: Proc. 39th Annual Symp. Foundations of Computer Science*, pages 534–543, 1998.

7 Amihood Amir, Dmitry Keselman, Gad M. Landau, Moshe Lewenstein, Noa Lewenstein, and Michael Rodeh. Text indexing and dictionary matching with one error. *J. Algorithms*, 37(2):309–325, 2000. doi:10.1006/jagm.2000.1104.

8 Amihood Amir, Gad M. Landau, Moshe Lewenstein, and Dina Sokol. Dynamic text and static pattern matching. *ACM Trans. Algorithms*, 3(2):19, 2007. doi:10.1145/1240233.1240242.

9 Amihood Amir, Avivit Levy, Ely Porat, and B. Riva Shalom. Dictionary matching with a few gaps. *Theor. Comput. Sci.*, 589:34–46, 2015. doi:10.1016/j.tcs.2015.04.011.

10 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58. ACM, 2015. doi:10.1145/2746539.2746612.

11 Ricardo Baeza-Yates, Gonzalo Navarro, Errki Sutinen, and Jorma Tarhio. Indexing methods for approximate text retrieval. Technical report, Dept. of CS, Univ. of Chile, March 1997.

12 Leonid Boytsov. Indexing methods for approximate dictionary searching: Comparative analysis. *ACM Journal of Experimental Algorithmics*, 16(1), 2011. doi:10.1145/1963190.1963191.

13 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *FOCS '14: Proc. 55th Annual Symp. Foundations of Computer Science*, pages 661–670, 2014.

14 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *FOCS '15: Proc. 56th Annual Symp. Foundations of Computer Science*, pages 79–97, 2015.

15 Peter Clifford and Raphaël Clifford. Simple deterministic wildcard matching. *Inf. Process. Lett.*, 101(2):53–54, 2007. doi:10.1016/j.ipl.2006.08.002.

16 Raphaël Clifford, Klim Efremenko, Benny Porat, and Ely Porat. A black box for online approximate pattern matching. *Inf. Comput.*, 209(4):731–736, 2011. doi:10.1016/j.ic.2010.12.007.

17 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 91–100. ACM, 2004. doi:10.1145/1007352.1007374.

18 Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 592–601. ACM, 2002. doi:10.1145/509907.509992.

19 Paweł Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Łącki, and Piotr Sankowski. Optimal dynamic strings. In *SODA '18: Proc. of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2018.

20 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.

21 Trinh N. D. Huynh, Wing-Kai Hon, Tak Wah Lam, and Wing-Kin Sung. Approximate string matching using compressed suffix arrays. *Theor. Comput. Sci.*, 352(1-3):240–249, 2006. doi:10.1016/j.tcs.2005.11.022.

22 Daniel M. Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *J. ACM*, 61(1):4:1–4:23, 2014. doi:10.1145/2559902.

23 Howard J. Karloff. Fast algorithms for approximately counting mismatches. *Inf. Process. Lett.*, 48(2):53–60, 1993. doi:10.1016/0020-0190(93)90177-B.

24 S. Rao Kosaraju. Efficient string matching. Manuscript, 1987.

25 Kasper Green Larsen. The cell probe complexity of dynamic range counting. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 85–94. ACM, 2012. doi:10.1145/2213977.2213987.

26 Kasper Green Larsen, Omri Weinstein, and Huacheng Yu. Crossing the logarithmic barrier for dynamic boolean data structure lower bounds. *arXiv preprint arXiv:1703.03575*, 2017.

27 Kasper Green Larsen and Ryan Williams. Faster online matrix-vector multiplication. In *SODA '17: Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2182–2189, 2017.

28 Mihai Pătraşcu and Erik D. Demaine. Tight bounds for the partial-sums problem. In *SODA '04: Proc. 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 20–29, 2004.

Lower Bounds for Combinatorial Algorithms for Boolean Matrix Multiplication

Debarati Das
Computer Science Institute of Charles University, Malostranské náměstí 25, 11800 Praha 1, Czech Republic
debaratix710@gmail.com

Michal Koucký
Computer Science Institute of Charles University, Malostranské náměstí 25, 11800 Praha 1, Czech Republic
koucky@iuuk.mff.cuni.cz

Michael Saks
Department of Mathematics, Rutgers University, Piscataway, NJ, USA
msaks30@gmail.com

Abstract

In this paper we propose models of combinatorial algorithms for the Boolean Matrix Multiplication (BMM), and prove lower bounds on computing BMM in these models. First, we give a relatively relaxed combinatorial model which is an extension of the model by Angluin (1976), and we prove that the time required by any algorithm for the BMM is at least $\Omega(n^3/2^{O(\sqrt{\log n})})$. Subsequently, we propose a more general model capable of simulating the "Four Russian Algorithm". We prove a lower bound of $\Omega(n^{7/3}/2^{O(\sqrt{\log n})})$ for the BMM under this model. We use a special class of graphs, called (r,t)-graphs, originally discovered by Rusza and Szemeredi (1978), along with randomization, to construct matrices that are hard instances for our combinatorial models.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis, Theory of computation → Abstract machines

Keywords and phrases Lower Bounds, Combinatorial Algorithm, Boolean Matrix Multiplication

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.23

Related Version The full version of the paper is available at `https://arxiv.org/abs/1801.05202`.

Funding The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement no. 616787. The second author was also partially supported by the Center of Excellence CE-ITI under the grant P202/12/G061 of GA ČR. The third author was supported in part by the Simons Foundation under Award 332622.

1 Introduction

Boolean matrix multiplication (BMM) is one of the core problems in discrete algorithms, with numerous applications including triangle detection in graphs [9], context-free grammar parsing [14], and transitive closure etc. [6, 7, 10]. Boolean matrix multiplication can be naturally interpreted as a path problem in graphs. Given a layered graph with three layers A, B, C and edges between layers A and B and between B and C, compute the bipartite graph between A and C in which $a \in A$ and $c \in C$ are joined if and only if they have a

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 23; pp. 23:1–23:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


common neighbor. If we identify the bipartite graph between A and B with its $A \times B$ boolean adjacency matrix $\mathcal{P}$ and the graph between B and C with its $B \times C$ boolean adjacency matrix $\mathcal{Q}$ then the desired graph between A and C is just the boolean product $\mathcal{P} \times \mathcal{Q}$.

Boolean matrix multiplication is the combinatorial counterpart of integer matrix multiplication. Both involve the computation of n^2 output values, each of which can be computed in a straightforward way in time $O(n)$ yielding a $O(n^3)$ algorithm for both problems. One of the celebrated classical results in algorithms is Strassen's discovery [12] that by ordinary matrix multiplication has *truly subcubic* algorithms, i.e. algorithms that run in time $O(n^\omega)$ for some $\omega < 3$, which compute the n^2 entries by computing and combining carefully chosen (and highly non-obvious) polynomial functions of the matrix entries. Subsequent improvements [5, 15, 8] have reduced the value of ω.

One of the fascinating aspects of BMM is that, despite its intrinsic combinatorial nature, the asymptotically fastest algorithm known is obtained by treating the boolean entries as integers and applying fast integer matrix multiplication. The intermediate calculations done for this algorithm seemingly have little to do with the combinatorial structure of the underlying bipartite graphs. There has been considerable interest in developing "combinatorial" algorithms for BMM, that is algorithms where the intermediate computations all have a natural combinatorial interpretation in terms of the original problem. Such interest is motivated both by intellectual curiosity, and by the fact that the fast integer multiplication algorithms are impractical because the constant factor hidden in $O(\cdot)$ is so large.

The straightforward n^3 algorithm has a straightforward combinatorial interpretation: for each pair of vertices a, c check each vertex of B to see whether it is adjacent to both a and c. The so-called Four Russian Algorithm by Arlazarov, Dinic, Kronrod, Faradzhev [13] solves BMM in $O(n^3/\log^2(n))$ operations, and was the first combinatorial algorithm for BMM with complexity $o(n^3)$. Overt the past 10 years, there have been a sequence of combinatorial algorithms [3, 4, 17] developed for BMM, all having complexities of the form $O(n^3/(\log n)^c)$ for increasingly large constants c. The best and most recent of these, due to Yu [17] has complexity $\hat{O}(n^3/\log^4 n)$ (where the $\hat{O}$ notation suppresses $poly(\log\log(n))$ factors. (It should be noted that the algorithm presented in each of these recent papers is for the problem of determining whether a given graph has a triangle; it was shown in [16] that a (combinatorial) algorithm for triangle finding with complexity $O(n^3/\log^c n)$ can be used as a subroutine to give a (combinatorial) algorithm for BMM with a similar complexity.)

While each of these combinatorial algorithms uses interesting and non-trivial ideas, each one saves only a polylogarithmic factor as compared to the straightforward algorithm, in contrast with the algebraic algorithms which save a power of n. The motivating question for the investigations in this paper is: Is there a truly subcubic combinatorial algorithm for BMM? We suspect that the answer is no.

In order to consider this question precisely, one needs to first make precise the notion of a combinatorial algorithm. This itself is challenging. To formalize the notion of a combinatorial algorithm requires some computation model which specifies what the algorithm states are, what operations can be performed, and what the cost of those operations is. If one examines each of these algorithms one sees that the common feature is that the intermediate information stored by the algorithm is of one of the following three types (1): for some pair of subsets (X, Y) with $X \subseteq A$ and $Y \subseteq B$, the submatrix (bipartite subgraph) induced by $\mathcal{P}$ on $X \times Y$ has some specified monotone property (such as, every vertex in X has a neighbor in Y), (2) for some pair of subsets (Y, Z) with $Y \subseteq B$ and $Z \subseteq C$, the bipartite subgraph induced by $\mathcal{Q}$ on $Y \times Z$ has some specific monotone property, or (3) for some pair of subsets (X, Z) with $X \subseteq A$ and $Z \subseteq C$, the bipartite subgraph induced by $\mathcal{P} \times \mathcal{Q}$ on $X \times Z$ has some specific monotone property.

If one accepts the above characterization of the possible information stored by the algorithm, we are still left with the problem of specifying the elementary steps that the algorithm is permitted to make to generate new pieces of information, and what the computational cost is. The goal in doing this is that the allowed operations and cost function should be such that they accurately reflect the cost of operations in an algorithm. In particular, we would like that our model is powerful enough to be able to simulate all of the known combinatorial algorithms with running time no larger than their actual running time, but not so powerful that it allows for fast (e.g. quadratic time) algorithms that are not implementable on a real computer. We still don't have a satisfactory model with these properties.

This paper takes a step in this direction. We develop a model which captures some of what a combinatorial algorithm might do. In particular our model is capable of efficiently simulating the Four Russian algorithm, but is sufficiently more general. We then prove a superquadratic lower bound in the model: Any algorithm for BMM in this model requires time at least $\Omega(n^{7/3}/2^{O(\sqrt{\log n})})$.

Unfortunately, our model is not strong enough to simulate the more recent combinatorial approaches. Our hope is that our approach provides a starting point for a more comprehensive analysis of the limitation of combinatorial algorithms.

One of the key features of our lower bound is the identification of a family of "hard instances" for BMM. In particular, we use tripartite graphs on roughly $3n$ vertices that have almost quadratic number a pairs of vertices from the first and the last layers connected by a single (*unique*) path via the middle layer. These graphs are derived from (r, t)-*graphs* of Rusza and Szemeredi [11], which are dense bipartite graphs on $2n$ vertices that can be decomposed into linear number of disjoint induced matchings. More recently, Alon, Moitra Sudakov [1] provides strengthening of Rusza and Szemeredi's construction although they lose in the parameters that are most relevant for us.

1.1 Combinatorial models

The first combinatorial model for BMM was given by Angluin [2]. For the product of $\mathcal{P} \times \mathcal{Q}$, the model allows to take bit-wise OR (*union*) of rows of the matrix $\mathcal{Q}$ to compute the individual rows of the resulting matrix $\mathcal{P}\mathcal{Q}$. The cost in this model is the number of unions taken. By a counting argument, Angluin [2] shows that there are matrices $\mathcal{P}$ and $\mathcal{Q}$ such that the number of unions taken must be $\Omega(n^2/\log n)$. This matches the number of unions taken by the Four Russian Algorithm, and in that sense the Four Russian Algorithm is optimal.

If the cost of taking each row union were counted as n, the total cost would become $\Theta(n^3/\log n)$. The Four Russian Algorithm improves this time to $O(n^3/\log^2 n)$ by leveraging "word-level parallelism" to compute each row union in time $O(n/\log n)$.

A possible approach to speed-up the Four Russian Algorithm would be to lower the cost of each union operation even further. The above analysis ignores the fact that we might be taking the union of rows with identical content multiple times. For example if $\mathcal{P}$ and $\mathcal{Q}$ are random matrices (as in the lower bound of Angluin) then each row of the resulting product is an all-one row. Such rows will appear after taking an union of merely $O(\log n)$ rows from $\mathcal{Q}$. An entirely naive algorithm would be taking unions of an all-one row with n possible rows of $\mathcal{Q}$ after only few unions. Hence, there would be only $O(n \log n)$ different unions to take for the total cost of $O(n^2 \cdot poly(\log n))$. We could quickly detect repetitions of unions by maintaining a short fingerprint for each row evaluated.

Our first model takes repetitions into account. Similarly to Angluin, we focus on the number of unions taken by the algorithm but we charge for each union differently. The natural cost of a union of rows with values $u, v \in \{0, 1\}^n$ counts the cost as the minimum

of the number of ones in u and v. This is the cost we count as one could use sparse set representation for u and v. In addition to that if unions of the same rows (vectors) are taken multiple times we charge all of them only ones, resp. we charge the first one the proper cost and all the additional unions are for a unit cost. As we have argued, on random matrices $\mathcal{P}$ and $\mathcal{Q}$, BMM will cost $O(n^2 \log n)$ in this model. Our first lower bound shows that even in this model, there are matrices for which the cost of BMM is almost cubic.

▶ **Theorem 1** (Informal statement). *In the row-union model with removed repetitions the cost of Boolean matrix multiplication is* $\Omega(n^3/2^{O(\sqrt{\log n})})$.

The next natural operation one might allow to the algorithm is to divide rows into pieces. This is indeed what the Four Russians Algorithm and many other algorithms do. In the Four Russian Algorithm, this corresponds to the "word-level parallelism". Hence we might allow the algorithm to break rows into pieces, take unions of the pieces, and concatenate the pieces back. In our more general model we set the cost of the partition and concatenation to be a unit cost, and we only allow to split a piece into continuous parts. More complex partitions can be simulated by performing many two-sided partitions and paying proportionally to the complexity of the partition. The cost of a union operation is again proportional to the smaller number of ones in the pieces, while repeated unions are charged for a unit cost. In this model one can implement the Four Russian Algorithm for the cost $O(n^3/\log^2 n)$, matching its usual cost. In the model without partitions the cost of the Four Russian Algorithm is $\Theta(n^3/\log n)$.

In this model we are able to prove super-quadratic lower bound when we restrict that all partitions happen first, then unions take place, and then concatenations.

▶ **Theorem 2** (Informal statement). *In the row-union model with partitioning and removed repetitions the cost of Boolean matrix multiplication is* $\Omega(n^{7/3}/2^{O(\sqrt{\log n})})$.

Perhaps, the characteristic property of "combinatorial" algorithms is that from the run of such an algorithm one can extract a combinatorial proof (*witness*) for the resulting product. This is how we interpret our models. For given $\mathcal{P}$ and $\mathcal{Q}$ we construct a witness circuit that mimics the work of the algorithm. The circuit operates on rows of $\mathcal{Q}$ to derive the rows of the resulting matrix $\mathcal{PQ}$. The values flowing through the circuit are bit-vectors representing the values of rows together with information on which union of which submatrix of $\mathcal{Q}$ the row represents. The gates can partition the vectors in pieces, concatenate them and take their union. For our lower bound we require that unions take place only after all partitions and before all concatenations. This seems to be a reasonable restriction since we do not have to emulate the run of an algorithm step by step but rather see what it eventually produces. Also allowing to mix partitions, unions and concatenations in arbitrary order could perhaps lead to only quadratic cost on all matrices. We are not able to argue otherwise.

The proper modeling of combinatorial algorithms is a significant issue here: one wants a model that is strong enough to capture known algorithms (and other conceivable algorithms) but not so strong that it admits unrealistic quadratic algorithms. We do not know how to do this yet, and the present paper is intended as a first step.

1.2 Our techniques

Central to our lower bounds are graphs derived from (r, t)-graphs of Rusza and Szemeredi [11]. Our graphs are tripartite with vertices split into parts A, B, C, where $|A| = |C| = n$ and $|B| = n/3$. The key property of these graphs is that there are almost quadratically many pairs $(a, c) \in A \times C$ that are connected via a single (*unique*) vertex from B. In terms of the

corresponding matrices $\mathcal{P}$ and $\mathcal{Q}$ this means that in order to evaluate a particular row of their product we must take a union of very specific rows in $\mathcal{Q}$. The number of rows in the union must be almost linear. Since $\mathcal{Q}$ is dense this might lead to an almost cubic cost for the whole algorithm provided different vertices in A are connected to different vertices in B so we take different unions.

This is not apriori the case for the (r, t)-derived graph but we can easily achieve it by removing edges between A and B at random, each independently with probability $1/2$. The neighborhoods of different vertices in A will be very different then. We call such a graph *diverse* (see a later section for a precise definition). It turns out that for our lower bound we need a slightly stronger property, not only that we take unions of different rows of $\mathcal{Q}$ but also that the results of these unions are different. We call this stronger property *unhelpfulness*.

Using unhelpfulness of graphs we are able to derive the almost cubic lower bound on the simpler model. Unhelpfulness is a much more subtle property than diversity, and we crucially depend on the properties of our graphs to derive it.

Next we tackle the issue of lower bounds for the partition model. This turns out to be a substantially harder problem, and most of the proof is deferred to the full version of this paper. One needs unhelpfulness on different pieces of rows (restrictions to columns of $\mathcal{Q}$), that is making sure that the result of union of some pieces does not appear (too often) as a result of union of another pieces. This is impossible to achieve in full generality. Roughly speaking what we can achieve is that different parts of *any* witness circuit cannot produce the same results of unions.

The key lemma that formalizes it (Lemma 11) shows that the results of unions obtained for a particular interval of columns in $\mathcal{Q}$ can be used at most $O(\log n)$ times on average in the rest of the circuit. This is a property of the graph which we refer to as that the graph *admitting only limited reuse*. This key lemma is technically complicated and challenging to prove (albeit elementary). Putting all the pieces together turns out to be also quite technical. We provide extensive overview in Section 4.

2 Notation and preliminaries

For any integer $k \geq 1$, $[k] = \{1, \ldots, k\}$. For a vertex a in a graph G and a subset S of vertices of G, $\Gamma(a)$ are the neighbors of a in G, and $\Gamma_S(a) = \Gamma(a) \cap S$. (To emphasize which graph G we mean we may write $\Gamma_{S,G}(a)$.) A *subinterval* of $C = \{c_1, c_2, \ldots, c_n\}$ is any set $K = \{c_i, c_{i+1}, \ldots, c_j\}$, for some $1 \leq i \leq j \leq |C|$. By $\min K$ we understand i and by $\max K$ we mean j. For a subinterval $K = \{c_i, c_{i+1}, \ldots, c_{i+\ell-1}\}$ of C and a vector $v \in \{0,1\}^\ell$, $K \restriction_v$ denotes the set $\{c_j \in K;\ v_{j-i+1} = 1\}$. For a vector $v \in \{0,1\}^n$, $v \restriction_K = v_i, v_{i+1}, \ldots, v_{i+\ell-1}$. For a binary vector v, $|v|$ denotes the number of ones in v.

2.1 Matrices

We will denote matrices by calligraphic letters $\mathcal{P}, \mathcal{Q}, \mathcal{R}$. All matrices we consider are binary matrices. For integers i, j, $\mathcal{P}_i$ is the i-th row of $\mathcal{P}$ and $\mathcal{P}_{i,j}$ is the (i,j)-th entry of $\mathcal{P}$. Let $\mathcal{P}$ be an $n_A \times n_B$ matrix and $\mathcal{Q}$ be an $n_B \times n_C$ matrix, for some integers n_A, n_B, n_C. We associate matrices $\mathcal{P}, \mathcal{Q}$ with a tripartite graph G. The vertices of G is the set $A \cup B \cup C$ where $A = \{a_1, \ldots, a_{n_A}\}$, $B = \{b_1, \ldots, b_{n_B}\}$ and $C = \{c_1, \ldots, c_{n_C}\}$. The edges of G are (a_i, b_k) for each i, k such that $\mathcal{P}_{i,k} = 1$, and (b_k, c_j) for each k, j such that $\mathcal{Q}_{k,j} = 1$. In this paper we only consider graphs of this form. Sometimes we may abuse notation and index matrix $\mathcal{P}$ by vertices of A and B, and similarly $\mathcal{Q}$ by vertices from B and C. For a set of indices $S \subseteq B$, $row(\mathcal{Q}_S) = \bigvee_{i \in S} \mathcal{Q}_i$ is the bit-wise Or of rows of $\mathcal{Q}$ given by S.

2.2 Model

Circuit. A *circuit* is a directed acyclic graph W where each node (*gate*) has in-degree either zero, one or two. The *degree* of a gate is its in-degree, the *fan-out* is its out-degree. Degree one gates are called *unary* and degree two gates are *binary*. Degree zero gates are called *input gates*. For each binary gate g, $\mathit{left}(g)$ and $\mathit{right}(g)$ are its two predecessor gates. A computation of a circuit proceeds by passing values along edges, where each gate processes its incoming values to decide on the value passed along the outgoing edges. The input gates have some predetermined values. The output of the circuit is the output value of some designated vertex or vertices.

Witness. Let $\mathcal{P}$ and $\mathcal{Q}$ be matrices of dimension $n_A \times n_B$ and $n_B \times n_C$, resp., with its associated graph G. A *witness* for the matrix product $\mathcal{P} \times \mathcal{Q}$ is a circuit consisting of input gates, unary *partition gates*, binary union gates and binary *concatenation gates*. The values passed along the edges are triples (S, K, v), where $S \subseteq B$ identifies a set of rows of the matrix $\mathcal{Q}$, the subinterval $K \subseteq C$ identifies a set of columns of $\mathcal{Q}$, and v is the restriction $\mathit{row}(\mathcal{Q}_S) \restriction_K$ of $\mathit{row}(\mathcal{Q}_S)$ to the columns of K. Each input gate outputs $(\{b\}, C, Q_b)$ for some assigned $b \in B$. A partition gate with an assigned subinterval $K' \subseteq C$ on input (S, K, v) outputs *undefined* if $K' \not\subseteq K$ and outputs (S, K', v') otherwise, where $v' \in \{0,1\}^{|K'|}$ is such that for each $j \in [|K'|]$, $v'_j = v_{j+\min K' - \min K}$. A union gate on inputs $(S_\mathrm{L}, K_\mathrm{L}, v_\mathrm{L})$ and $(S_\mathrm{R}, K_\mathrm{R}, v_\mathrm{R})$ from its children outputs *undefined* if $K_\mathrm{L} \neq K_\mathrm{R}$, and outputs $(S_\mathrm{L} \cup S_\mathrm{R}, K_\mathrm{L}, v_\mathrm{L} \cup v_\mathrm{R})$ otherwise. A concatenation gate, on inputs $(S_\mathrm{L}, K_\mathrm{L}, v_\mathrm{L})$ and $(S_\mathrm{R}, K_\mathrm{R}, v_\mathrm{R})$ where $\min K_\mathrm{L} \leq \min K_\mathrm{R}$, is *undefined* if $\max K_\mathrm{L} + 1 < \min K_\mathrm{R}$ or $S_\mathrm{L} \neq S_\mathrm{R}$ or $\max K_\mathrm{L} > \max K_R$ and outputs $(S_\mathrm{L}, K_\mathrm{L} \cup K_\mathrm{R}, v')$ otherwise, where v' is obtained by concatenating v_L with the last $(\max K_\mathrm{R} - \max K_\mathrm{L})$ bits of v_R.

It is straightforward that whether a gate is undefined depends solely on the structure of the circuit but not on the actual values of $\mathcal{P}$ or $\mathcal{Q}$. We will say that the circuit is *structured* if union gates do not send values into partition gates, and concatenation gates do not send values into partition and union gates. Such a circuit first breaks rows of $\mathcal{Q}$ into parts, computes union of compatible parts and then assembles resulting rows using concatenation.

We say that a witness W is a *correct witness* for $\mathcal{P} \times \mathcal{Q}$ if W is structured, no gate has undefined output, and for each $a \in A$, there is a gate in W with output $(\Gamma_B(a), C, v)$ for $v = \mathit{row}(\mathcal{Q}_{\Gamma_B(a)})$.

Cost. The *cost* of the witness W is defined as follows. For each union gate g with inputs $(S_\mathrm{L}, K_\mathrm{L}, v_\mathrm{L})$ and $(S_\mathrm{R}, K_\mathrm{R}, v_\mathrm{R})$ and an output (S, K, v) we define its *row-class* to be $\mathit{class}(g) = \{v, v_\mathrm{L}, v_\mathrm{R}\}$. If T is a set of union gates from W, $\mathit{class}(T) = \{\{u, v, z\}, \{u, v, z\}$ is the row-class of some gate in $T\}$. The cost of a row-class $\{u, v, z\}$ is $\mathit{min}\{|u|, |v|, |z|\}$. The cost of T is $\sum_{\{u,v,z\} \in \mathit{class}(T)} \mathit{min}\{|u|, |v|, |z|\}$. The *cost of witness* W is the number of gates in W plus the cost of the set of all union gates in W.

We can make the following simple observation.

▶ **Proposition 3.** *If W is a correct witness for $\mathcal{P} \times \mathcal{Q}$, then for each $a \in A$, there exists a collection of subintervals $K_1, \ldots, K_\ell \subseteq C$ such that $C = \bigcup_i K_i$ and for each $i \in [\ell]$, there is a union gate in W which outputs $(\Gamma_B(a), K_i, \mathit{row}(\mathcal{Q}_{\Gamma_B(a)}) \restriction_{K_i})$.*

Union and resultant circuit. One can look at the witness circuit from two separate angles which are captured in the next definitions. A *union circuit over a universe B* is a circuit with gates of degree zero and two where each gate g is associated with a subset $\mathit{set}(g)$ of B so that

for each binary gate g, $set(g) = set(left(g))) \cup set(right(g))$. For integer $\ell \geq 1$, a *resultant circuit* is a circuit with gates of degree zero and two where each gate g is associated with a vector $row(g)$ from $\{0,1\}^\ell$ so that for each binary gate g, $row(g) = row(left(g)) \vee row(right(g))$, where $\vee$ is a coordinate-wise OR.

For a vertex $a \in A$ and a subinterval $K = \{c_i, c_{i+1}, \ldots, c_{i+\ell-1}\}$ of C, a *union witness* for (a, K) is a union circuit W over B with a single output gate g_{out} where $set(g_{\text{out}}) = \Gamma_B(a)$ and for each input gate g of W, $set(g) = \{b\}$ for some $b \in B$ connected to a.

Induced union witness. Let W be a correct witness for $\mathcal{P} \times \mathcal{Q}$. Pick $a \in A$ and a subinterval $K \subseteq C$. Let there be a union gate g in W with output $(\Gamma_B(a), K, row(\mathcal{Q}_{\Gamma_B(a)}) \restriction_K)$. An *induced union witness* for (a, K) is a union circuit over B whose underlying graph consists of copies of the union gates that are predecessors of g, and a new input gate for each input or partition gate that feeds into one of the union gates. They are connected in the same way as in W. For each gate g in the induced witness we let $set(g) = S$ whenever its corresponding gate in W outputs (S, K', v) for some K' and v. From the correctness of W it follows that each such $K' = K$ and the resulting circuit is a correct union witness for (a, K).

2.3 (r, t)-graphs

We will use special type of graphs for constructing matrices which are hard for our combinatorial model of Boolean matrix multiplication. For integers $r, t \geq 1$, an (r, t)-graph is a graph whose edges can be partitioned into t pairwise disjoint induced matchings of size r. Somewhat counter-intuitively as shown by Rusza and Szemeredi [11] there are dense graphs on n vertices that are (r, t)-graphs for r and t close to n.

▶ **Theorem 4** (Rusza and Szemerédi [11]). *For all large enough integers n, for $\delta_n = 1/2^{\Theta(\sqrt{\log n})}$ there is a $(\delta_n n, n/3)$-graph $G_n^{r,t}$.*

A more recent work of Alon, Moitra Sudakov [1] provides a construction of a (r, t)-graphs on n vertices with $rt = (1 - o(1))\binom{n}{2}$ and $r = n^{1-o(1)}$. The graphs of Rusza and Szemerédi are sufficient for us.

Let $G_n^{r,t}$ be the graph from the previous theorem and let $M_1, M_2, \ldots, M_{n/3}$ be the disjoint induced matchings of size $\delta_n n$. We define a tripartite graph G_n as follows: G_n has vertices $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_{n/3}\}$ and $C = \{c_1, \ldots, c_n\}$. For each i, j, k such that $(i, j) \in M_k$ there are edges (a_i, b_k) and (b_k, c_j) in G_n. The following immediate lemma states one of the key properties of G_n.

▶ **Lemma 5.** *If $(i, j) \in M_k$ in $G_n^{r,t}$ then there is a unique path between a_i and c_j in G_n.*

For the rest of the paper, we will fix the graphs G_n. Additionally, we will also use a graph $\widetilde{G}_n$ which is obtained from G by removing each edge between A and B independently at random with probability $1/2$. (Technically, $\widetilde{G}_n$ is a random variable.) When n is clear from the context we will drop the subscript n.

Fix some large enough n. Let $\mathcal{P}$ be the $n \times n/3$ adjaceny matrix between A and B in G and $\mathcal{Q}$ be the $n/3 \times n$ adjacency matrix between B and C in G. The adjacency matrix between A and B in $\widetilde{G}$ will be denoted by $\widetilde{\mathcal{P}}$. ($\widetilde{\mathcal{P}}$ is also a random variable.) The adjacency matrix between B and C in $\widetilde{G}$ is $\mathcal{Q}$.

We say that c is *unique* for $a \in A$ if there is exactly one $b \in B$ such that (a, b) and (b, c) are edges in G. The previous lemma implies that on average a has many unique vertices c in G_n, namely $\delta_n n/3$. For $S \subseteq C$, let $S[a]$ denote the set of vertices from S that are unique for

a in G. E.g., $C[a]$ are all vertices unique for a. Let $\beta_a(S)$ denote the set of vertices from B that are connected to a and some vertex in $S[a]$. Notice, $|\beta_a(S)| = |S[a]|$. Since $\beta_a(\cdot)$ and $\cdot[a]$ depend on edges in graph G, to emphasise which graph we have in mind we may subscript them by G: $\beta_{a,G}(\cdot)$ and $\cdot[a]_G$.

For the randomized graph $\widetilde{G}$ we will denote by $S[a]'_{\widetilde{G}}$ the set of vertices from S that are unique for a in G and that are connected to a via B also in $\widetilde{G}$. (Thus, vertices from S that are not unique for a in G but became unique for a in $\widetilde{G}$ are not included in $S[a]'_{\widetilde{G}}$.) Let $\beta'_{a,\widetilde{G}}(S)$ denotes $\beta_a(S[a]'_{\widetilde{G}})$

2.4 Diverse and unhelpful graphs

In this section we define two properties of $\widetilde{G}$ that capture the notion that one needs to compute many different unions of rows of $\mathcal{Q}$ to calculate $\widetilde{\mathcal{P}} \times \mathcal{Q}$. The simpler condition stipulates that neighborhoods of different vertices from A are quite different. The second condition stipulates that not only the neighborhoods of vertices from A are different but also the unions of rows from $\mathcal{Q}$ that correspond to these neighborhoods are different.

Let G_n and $\widetilde{G}_n$ and $\mathcal{P}, \mathcal{Q}, \widetilde{\mathcal{P}}$ be as in the previous section. For integers $k, \ell \geq 1$, we say $\widetilde{G}$ is (k,ℓ)-*diverse* if for every set $S \subseteq B$ of size at least ℓ, no k vertices in A are all connected to all the vertices of S.

▶ **Lemma 6.** *Let $c, d \geq 4$ be integers. The probability that $\widetilde{G}_n$ is $(c\log n, d\log n)$-diverse is at least $1 - n^{-(cd/2)\log n}$.*

Proof. Let $k = c\log n$ and $\ell = d\log n$. $\widetilde{G}$ is not (k,ℓ)-diverse if for some set $S \subseteq B$ of size ℓ, and some k-tuple of distinct vertices $a_1, \ldots, a_k \in A$, each vertex a_i is connected to all vertices from S in $\widetilde{G}$. The probability that all vertices of a given k-tuple $a_1, \ldots, a_k \in A$ are connected to all vertices in S in $\widetilde{G}$ is at most $2^{-k\ell}$. (The probability is zero if some a_i is not connected to some vertex from S in G.) Hence, the probability that there is some set $S \subseteq B$ of size ℓ, and some k-tuple of distinct vertices $a_1, \ldots, a_k \in A$ where each vertex a_i is connected to all vertices from S in $\widetilde{G}$ is bounded by:

$$\binom{n}{c\log n} \cdot \binom{n}{d\log n} \cdot 2^{-cd\log^2 n} \leq n^{(c+d)\log n} \cdot 2^{-cd\log^2 n} \leq \frac{1}{n^{(cd/2)\log n}}$$

where the second inequality follows from $c, d \geq 4$. ◀

For $S \subseteq B$, $a \in A$ and a subinterval $K \subseteq C$, we say that S is *helpful for a on K* if there exists a set $S' \subseteq \beta'_{a,\widetilde{G}}(K)$ such that $|S| \leq |S'|$ and $C[a]_G \cap (K \restriction_{row(\mathcal{Q}_S)}) = C[a]_G \cap (K \restriction_{row(\mathcal{Q}_{S'})})$. In other words, the condition means that $row(\mathcal{Q}_s)$ and $row(\mathcal{Q}_{S'})$ agree on coordinates in K that correspond to vertices unique for a in G. This is a necessary precondition for $row(\mathcal{Q}_S) \restriction_K = row(\mathcal{Q}_{S'}) \restriction_K$ which allows one to focus only on the *hard-core* formed by the unique vertices. In particular, if for some $S'' \subseteq \Gamma_{B,\widetilde{G}}(a)$ in $\widetilde{G}$, $row(\mathcal{Q}_S) \restriction_K = row(\mathcal{Q}_{S''}) \restriction_K$, then $S' = S'' \cap \beta'_{a,\widetilde{G}}(K)$ satisfies $C[a]_G \cap (K \restriction_{row(\mathcal{Q}_S)}) = C[a]_G \cap (K \restriction_{row(\mathcal{Q}_{S'})})$. (See the proof below.)

For integers $k, \ell \geq 1$, we say $\widetilde{G}$ is (k,ℓ)-*unhelpful on K* if for every set $S \subseteq B$ of size at least ℓ, there are at most k vertices in A for which S is helpful on K.

▶ **Lemma 7.** *Let $c, d \geq 4$ be integers. Let $K = \{c_i, c_{i+1}, \ldots, c_{i+\ell-1}\}$ be a subinterval of C. The probability that $\widetilde{G}_n$ is $(c\log n, d\log n)$-unhelpful on K is at least $1 - n^{-(cd/2)\log n}$.*

Proof. Take any set $S \subseteq B$ of size $\ell \geq d \log n$ and arbitrary vertices $a_1, \ldots, a_k \in A$ for $k = c \log n$. Consider $row(\mathcal{Q}_S) \restriction_K$ and some $i \in [k]$. Since edges between B and C are always the same in $\widetilde{G}$, $row(\mathcal{Q}_S) \restriction_K$ is always the same in $\widetilde{G}$. If S is helpful on K for a_i then there exists $S_i \subseteq \beta'_{a,\widetilde{G}}(K)$ such that $|S_i| \geq \ell$ and $C[a]_G \cap (K \restriction_{row(\mathcal{Q}_{S_i})}) = C[a]_G \cap (K \restriction_{row(\mathcal{Q}_S)})$. It turns out that given a_i, the possible S_i is uniquely determined by $row(\mathcal{Q}_S) \restriction_K$. Whenever $row(\mathcal{Q}_S) \restriction_K$ has one in a position c that corresponds to a unique vertex of a in G, $row(\mathcal{Q}_{S_i}) \restriction_K$ must have one there as well so the corresponding b must be in S_i. Conversely, whenever $row(\mathcal{Q}_S) \restriction_K$ has zero in a position c that corresponds to a unique vertex of a in G, $row(\mathcal{Q}_{S_i}) \restriction_K$ must have zero there as well so the corresponding b is not in S_i. The probability that $S_i \subseteq \beta'_{a,\widetilde{G}}(K)$ is $2^{-|S_i|}$.

Hence, the probability over choice of $\widetilde{G}$ that S is helpful for a_i on K is at most $2^{-\ell}$. For different a_i's this probability is independent as it only depends on edges between a_i and B. Thus the probability that S is helpful for $a_1, \ldots, a_k$ is at most $2^{-\ell k}$.

There are at most $\binom{n}{\ell} \cdot \binom{n}{k}$ choices for the set S of size ℓ and $a_1, \ldots, a_k$. Hence, the probability that $\widetilde{G}$ is not $(c \log n, d \log n)$-unhelpful on K is at most:

$$\begin{aligned}
\sum_{\ell=d\log n}^{n} \binom{n}{\ell} \cdot \binom{n}{k} \cdot 2^{-\ell k} &\leq \sum_{\ell=d\log n}^{n} n^{\ell} \cdot n^{k} \cdot 2^{-\ell k} \\
&\leq \sum_{\ell=d\log n}^{n} 2^{(\ell+k)\log n - \ell k} \\
&\leq \sum_{\ell=d\log n}^{n} 2^{-\ell k/2} \\
&\leq \sum_{\ell=d\log n}^{n} \frac{1}{n^{(cd/2)\log n}}
\end{aligned}$$

where the third inequality follows from $c, d \geq 4$. ◀

3 Union circuits

Our goal is to prove the following theorem:

▶ **Theorem 8.** *There is a constant $c > 0$ such that for all n large enough there are matrices $\mathcal{P} \in \{0,1\}^{n \times n/3}$ and $\mathcal{Q} \in \{0,1\}^{n/3 \times n}$ such that any correct witness for $\mathcal{P} \times \mathcal{Q}$ consisting of only union gates has cost at least $n^3/2^{c\sqrt{\log n}}$.*

Here by consisting of only union gates we mean consisting of union gates and input gates. Our almost cubic lower bound on the cost of union witnesses is an easy corollary to the following lemma.

▶ **Lemma 9.** *Let n be a large enough integer and $\widetilde{G}_n$ be the graph from Section 2.3, and $\widetilde{\mathcal{P}}, \mathcal{Q}$ be its corresponding matrices. Let W be a correct witness for $\widetilde{\mathcal{P}} \times Q$ consisting of only union gates. Let $\widetilde{\mathcal{P}}$ have at least m ones. Let each row of $\mathcal{Q}$ have at least r ones. If $\widetilde{G}$ is (k,ℓ)-unhelpful on C for some integers $k, \ell \geq 1$ then any correct witness for $\widetilde{\mathcal{P}} \times \mathcal{Q}$ consisting of only union gates has cost at least $(mr/2k\ell) - nr/k$.*

Proof. Let W be a correct witness for $\widetilde{\mathcal{P}} \times \mathcal{Q}$ consisting of only union gates. For each gate g of W with output (S, C, v), for some v, define $set(g) = S$. Consider $a \in A$. Let g_a be a gate of W such that $set(g_a) = \Gamma_{B,\widetilde{G}}(a)$ (which equals $\beta'_{a,\widetilde{G}}(C)$). Take a maximal

set D_a of gates from W, descendants of g_a, such that for each $g \in D_a$, $|set(g)| \geq \ell$ and either $|set(left(g))| < \ell$ or $|set(right(g))| < \ell$, and furthermore for $g \neq g' \in D_a$, $\{set(g), set(left(g)), set(right(g))\} \neq \{set(g'), set(left(g')), set(right(g'))\}$.

Notice, if $g \neq g' \in D_a$ then $class(g) \neq class(g')$. This is because for any sets $S \neq S' \subseteq set(g_a)$, $row(\mathcal{Q}_S) \neq row(\mathcal{Q}_{S'})$. (Say, $b \in S \setminus S'$, then there is 1 in $\mathcal{Q}_b$ which corresponds to a vertex c unique for a. Thus, $row(\mathcal{Q}_S)_c = 1$ whereas $row(\mathcal{Q}_{S'})_c = 0$.)

We claim that since D_a is maximal, $|D_a| \geq \lfloor |set(g_a)|/2\ell \rfloor$. We prove the claim. Assume $set(g_a) \geq 2\ell$ otherwise there is nothing to prove. Take any $b \in set(g_a)$ and consider a path $g_0, g_1, \ldots, g_p = g_a$ of gates in W such that $set(g_0) = \{b\}$. Since $|set(g_0)| = 1$, $|set(g_a)| \geq 2\ell$ and $set(g_{i-1}) \subseteq set(g_i)$, there is some g_i with $|set(g_i)| \geq \ell$ and $|set(g_{i-1})| < \ell$. By maximality of D_a there is some gate $g \in D_a$ such that $\{set(g), set(left(g)), set(right(g))\} = \{set(g_i), set(left(g_i)), set(right(g_i))\}$. Hence, b is in $set(left(g))$ or $set(right(g))$ of size $< \ell$. Thus

$$set(g_a) \subseteq \bigcup_{g \in D_a;\ |set(left(g))| < \ell} set(left(g)) \cup \bigcup_{g \in D_a;\ |set(right(g))| < \ell} set(right(g))$$

Hence, $|set(g_a)| \leq 2\ell \cdot |D_a|$ and the claim follows.

For a given a, gates in D_a have different row-classes. Since $\widetilde{G}$ is (k, ℓ)-unhelpful on C, the same row-class can appear in D_a only for at most k different a's. (Say, there were $a_1, a_2, \ldots, a_{k+1}$ vertices in A and gates $g_1 \in D_{a_1}, \ldots, g_{k+1} \in D_{a_{k+1}}$ of the same row-class. For each $i \in [k+1]$, $set(g_i) \subseteq \Gamma_{B,\widetilde{G}}(a_i) = \beta'_{a_i,\widetilde{G}}(C)$ and $|set(g_i)| \geq \ell$. The smallest $set(g_i)$ would be helpful for $a_1, a_2, \ldots, a_{k+1}$ contradicting the unhelpfulness of $\widetilde{G}$.) Since

$$\sum_a |D_a| \geq \sum_a \lfloor |set(g_a)|/2\ell \rfloor \geq \frac{m}{2\ell} - n,$$

witness W contains gates of at least $(m/2k\ell) - n/k$ different row-classes. Since, each $\mathcal{Q}_b$ contains at least r ones, the total cost of W is as claimed. ◀

Proof of Theorem 8. Let $\widetilde{G}_n$ be the graph from Section 2.3, and $\widetilde{\mathcal{P}}, \mathcal{Q}$ be its corresponding matrices. Let $r = n\delta_n$. By Lemma 7, the graph $\widetilde{G}$ is $(5\log n, 5\log n)$-unhelpful on C with probability at least $1 - 1/n^{\log n}$, and by Chernoff bound, $\widetilde{\mathcal{P}}$ contains at least $nr/10$ ones with probability at least $1 - exp(n)$. So with probability at least $1/2$, $\widetilde{\mathcal{P}}$ has $m \geq nr/10$ ones while $\widetilde{G}$ is $(5\log n, 5\log n)$-unhelpful on C. By the previous lemma, any witness for $\widetilde{\mathcal{P}} \times \mathcal{Q}$ is of cost $(nr^2/25\log n) - nr/5\log n$. For large enough n, this is at least $nr^2/50\log n = n^3\delta_n^2/50\log n$, and the theorem follows. ◀

4 Circuits with partitions

In this section, our goal is to prove the lower bound $\Omega(n^{7/3}/2^{O(\sqrt{\log n})})$ on the cost of a witness for matrix product when the witness is allowed to partition the columns of $\mathcal{Q}$. Namely:

▶ **Theorem 10.** *For all n large enough there are matrices $\mathcal{P} \in \{0,1\}^{n \times n/3}$ and $\mathcal{Q} \in \{0,1\}^{n/3 \times n}$ such that any correct witness for $\mathcal{P} \times \mathcal{Q}$ has cost at least $\Omega(n^{7/3}/2^{O(\sqrt{\log n})})$.*

Due to space limitations we provide only an overview of the proof. The proof builds on ideas seen already in the previous part but also requires several additional ideas. Consider a correct witness for $\widetilde{\mathcal{P}} \times \mathcal{Q}$. We partition its union gates based on their corresponding subinterval of C. If there are many vertices in A that use many different subintervals (roughly

$\Omega(n^{4/3})$ in total) the lower bound follows by counting the total number of gates in the circuit using diversity of $\widetilde{G}$ (Lemma 13). If there are many vertices in A which use only few subintervals (less than roughly $O(n^{1/3})$ each) then these subintervals must be large on average (about $n^{2/3}$) and contain lots of vertices from C unique for their respective vertices from A.

In this case we divide the circuit (its union gates) based on their subinterval, and we calculate the contribution of each part separately. To do that we have to limit the amount of reuse of a given row-class within each part, and also among distinct parts. Within each part we limit the amount of reuse using a similar technique to Lemma 9 based on unhelpfulness of the graph (Lemma 12). However, for distinct parts we need a different tool which we call *limited reuse*. Limited reuse is somewhat different than unhelpfulness in the type of guarantee we get. It is a weaker guarantee as we are not able to limit the reuse of a row-class for each single gate but only the total reuse of row-classes of all the gates in a particular part. On average the reuse is again roughly $O(\log n)$.

However, the number of gates in a particular part of the circuit might be considerably larger than the number of gates we are able to charge for work in that part. In general, we are only able to charge gates that already made some non-trivial progress in the computation (as otherwise the gates could be reused heavily.) We overcome this obstacle by balancing the size of the part against the number of *chargeable* gates in that part.

If the total number of gates in the part is at least $n^{1/3}$-times larger than the total number of chargeable gates, we charge the part for its size. Otherwise we charge it for work. Each chargeable gates contributes by about $n^{2/3}$ units of work or more, however this can be reused almost $n^{1/3}$-times elsewhere. Either way, approximately $\Omega(n^{7/3})$ of work must be done in total. Now we present the actual proof.

The actual proof of the theorem builds on several key lemmas which we state next. Due to space limitations, details of the proof are deferred to the full version of this paper. In order to prove the theorem we need few more definitions. Let G_n and $\widetilde{G}_n$ and $\mathcal{P}, \mathcal{Q}, \widetilde{\mathcal{P}}$ be as in the Section 2.3. All witness circuits in this section are with respect to $\widetilde{\mathcal{P}} \times \mathcal{Q}$ (i.e., $\widetilde{G}_n$). Let c_0 and c_1 be some constants that we will fix later.

The following definition aims to separate contribution from different rows within a particular subcircuit. A witness circuit may benefit from taking a union of the same row of $\mathcal{Q}$ multiple times to obtain a particular union. This could help various gates to attain the same row-class. In order to analyze the cost of the witness we want to effectively prune the circuit so that contribution from each row of $\mathcal{Q}$ is counted at most once. The following definition captures this prunning.

Let W be a union circuit over B with a single vertex g_{out} of out-degree zero (*output gate*). The *trimming* of W is a map that associates to each gate g of W a subset $\mathit{trim}(g) \subseteq \mathit{set}(g)$ such that $\mathit{trim}(g_{\text{out}}) = \mathit{set}(g_{\text{out}})$ and for each non-input gate g, $\mathit{trim}(g) = \mathit{trim}(\mathit{left}(g)) \dot{\cup} \mathit{trim}(\mathit{right}(g))$. For each circuit W, we fix a canonical trimming that is obtained from $\mathit{set}(\cdot)$ by the following process: For each $b \in \mathit{set}(g_{\text{out}})$, find the left-most path from g_{out} to an input gate g such that $b \in \mathit{set}(g)$, and remove b from $\mathit{set}(g')$ of every gate g' that is *not* on this path.

Given the trimming of a union circuit W we will focus our attention only on gates that contribute substantially to the cost of the computation. We call such gates *chargeable* in the next definition. For a vertex $a \in A$ and a subinterval $K \subseteq C$, let W be a union witness for (a, K) with its trimming. We say a gate g in W is (a, K)-*chargeable* if $|\mathit{trim}(g) \cap \beta'_{a,\widetilde{G}}(K)| \geq c_0 \log n$ and $\mathit{trim}(\mathit{left}(g)) \cap \beta'_{a,\widetilde{G}}(K)$ and $\mathit{trim}(\mathit{right}(g)) \cap \beta'_{a,\widetilde{G}}(K)$ are both different from $\mathit{trim}(g) \cap \beta'_{a,\widetilde{G}}(K)$. (a, K)-*Chargeable descendants of* g are (a, K)-chargeable gates g' in W

where $trim(g') \cap \beta'_{a,\widetilde{G}}(K) \subseteq trim(g) \cap \beta'_{a,\widetilde{G}}(K)$. Observe that the number of (a,K)-chargeable descendants of a gate g is at most $|trim(g) \cap \beta'_{a,\widetilde{G}}(K)| + 1 - c_0 \log n$.

From a correct witness for $\widetilde{\mathcal{P}} \times \mathcal{Q}$, we extract some induced union circuit W for (a,K) and some resultant circuit W'. We say that a gate g from W *is compatible* with a gate g' from W' if $row(\mathcal{Q}_{set(g)}) \restriction_K = row(g')$.

We want to argue that chargeable gates corresponding to gates of a given correct witness have many different row-classes. Hence, we want to bound the number of gates whose result is compatible with each other. This is akin to the notion of helpfulness. In the case of helpfulness we were able to limit the repetition of the same row-class for individual gates operating on the same subinterval of columns of $\mathcal{Q}$. In addition to that we need to limit the occurence of the same row-class for gates that operate on distinct subintervals. As opposed to the simpler case of helpfulness, we will need to focus on the global count of row-classes that can be reused elsewhere from gates operating on the same subinterval. The next definition encapsulates the desired property of $\widetilde{G}$.

For $a, a' \in A$ and subintervals K, K' of C, we say that (a,K) and (a',K') are *independent* if either $a \neq a'$ or $K \cap K' = \emptyset$. A resultant circuit W' over $\{0,1\}^\ell$ is consistent with $\mathcal{Q}$, if there exists a subinterval $K \subseteq C$ of size ℓ, such that for each input gate g of W', $row(g) = \mathcal{Q}_b \restriction_K$ for some $b \in B$. We say that $\widetilde{G}$ *admits only limited reuse* if for any resultant circuit W' of size at most n^3 which is consistent with $\mathcal{Q}$ and any correct witness circuit W for $\widetilde{\mathcal{P}} \times \mathcal{Q}$, the number of gates in any induced union witnesses $W_1, \ldots, W_s$ for any pairwise independent pairs $(a_1, K_1), \ldots, (a_s, K_s)$ that are chargeable and compatible with some gate in W' is at most $c_1 |W'| \log n$.

We will show that with high probability $\widetilde{G}$ admits only limited reuse.

▶ **Lemma 11.** *Let $c_1 \geq 7$ and $c_0 \geq 20$ be constants. Let n be a large enough integer. Let $\widetilde{G}_n$ be the graph from Section 2.3, and $\widetilde{\mathcal{P}}, \mathcal{Q}$ be its corresponding matrices. The probability that $\widetilde{G}$ admits only limited reuse is at least $1 - 1/n$.*

The next lemma lower bounds the contribution of chargeable gates to the total cost of the witness. It is similar in spirit to Lemma 9 and its proof is similar. It focuses on union gates dealing with a particular subinterval $K \subseteq C$.

▶ **Lemma 12** (Partition version)**.** *Let $\widetilde{G}, \widetilde{\mathcal{P}}, \mathcal{Q}, W$ be as above. Let $r, k > 1$ be integers and $\ell = c_0 \log n$. Let $K \subseteq C$ be a subinterval. Let $R \subseteq B$ be such that for each b in R, $\mathcal{Q}_b \restriction_K$ has at least r ones. Let $A' \subseteq A$ be such that for each $a \in A'$, $|R \cap \beta'_{a,\widetilde{G}}(K)| \geq 2\ell$. Let $m = \sum_{a \in A'} |R \cap \beta'_{a,\widetilde{G}}(K)|$. If $\widetilde{G}$ is (k,ℓ)-unhelpful on K then there is a set D of union gates in W such that*

1. *Each gate in D is (a,K)-chargeable for some vertex $a \in A$, and*
2. *The number of different row-classes of gates in D of cost $\geq r$ is at least $m/4k\ell$.*

If the witness for $\widetilde{\mathcal{P}} \times Q$ involves many subintervals for many vertices we will apply the next lemma. By Proposition 3 each $a \in A$ is associated with distinct subintervals $K_{a,1}, \ldots, K_{a,\ell_a} \subseteq C$, for some ℓ_a, such that $C = \bigcup_{j \in [\ell_a]} K_{a,j}$ and there are union gates $g_{a,1}, \ldots, g_{a,\ell_a}$ in W such that $g_{a,j}$ outputs $(\Gamma_{B,\widetilde{G}}(a), K_{a,j}, v_{a,j})$ for some $v_{a,j} \in \{0,1\}^{|K_{a,j}|}$.

▶ **Lemma 13.** *Let W, ℓ_a's, $K_{a,j}$'s, $g_{a,j}$'s be as above. Let $c, d \geq 4$ and $\ell, r \geq 1$ be integers where r is large enough. Let $L = \{a \in A, \ell_a \geq \ell \;\&\; |\Gamma_{B,\widetilde{G}}(a)| \geq r\}$. If $\widetilde{G}$ is $(c \log n, d \log n)$-diverse then the size of W is at least $r\ell \cdot |L|/(2cd \log^2 n)$.*

The proof of the main theorem that leverages these lemmas is given in the full version of the paper.

References

1 Noga Alon, Ankur Moitra, and Benny Sudakov. Nearly complete graphs decomposable into large induced matchings and their applications. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1079–1090. ACM, 2012. doi:10.1145/2213977.2214074.

2 Dana Angluin. The four russians' algorithm for boolean matrix multiplication is optimal in its class. In *ACM SIGACT News*, pages 19–33, 1976.

3 Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. *Theory of Computing*, 8(1):69–94, 2012. doi:10.4086/toc.2012.v008a004.

4 Timothy M. Chan. Speeding up the four russians algorithm by about one more logarithmic factor. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 212–217. SIAM, 2015. doi:10.1137/1.9781611973730.16.

5 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.

6 Michael J Fischer and Albert R Meyer. Boolean matrix multiplication and transitive closure. In *12th Annual Symposium on Switching and Automata Theory*, pages 129–131, 1971.

7 M. E. Furman. Application of a method of fast multiplication of matrices in the problem of finding the transitive closure of a graph. *Soviet Mathematics Doklady*, page 11(5):1252, 1970.

8 François Le Gall. Powers of tensors and fast matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303. ACM, 2014. doi:10.1145/2608628.2608664.

9 Alon Itai. Finding a minimum circuit in a graph. In John E. Hopcroft, Emily P. Friedman, and Michael A. Harrison, editors, *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 1–10. ACM, 1977. doi:10.1145/800105.803390.

10 Ian Munro. Efficient determination of the transitive closure of a directed graph. *Information Processing Letters*, pages 1(2):56—-58, 1971.

11 I. Ruszá and E. Szemerédi. Triple systems with no six points carrying three triangles. In *Colloquia Mathematica Societatis János Bolyai*, pages 939–945, 1978.

12 V. Strassen. Gaussian elimination is not optimal. In *Numer. Math*, pages 13:354—-356, 1969.

13 M. A. Kronrod V. Z. Arlazarov, E. A. Dinic. On economical construction of the transitive closure of a directed graph. *Soviet Mathematics Doklady*, pages 11(5):1209—-1210, 1970.

14 Leslie G. Valiant. General context-free recognition in less than cubic time. *Journal of computer and system sciences*, pages 10(2):308—-315, 1975.

15 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898. ACM, 2012. doi:10.1145/2213977.2214056.

16 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 645–654. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.67.

17 Huacheng Yu. An improved combinatorial algorithm for boolean matrix multiplication. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015,*

Kyoto, Japan, July 6-10, 2015, Proceedings, Part I, volume 9134 of *Lecture Notes in Computer Science*, pages 1094–1105. Springer, 2015. doi:10.1007/978-3-662-47672-7_89.

Solving the Rubik's Cube Optimally is NP-complete

Erik D. Demaine
MIT Computer Science and Artificial Intelligence Laboratory,
32 Vassar St., Cambridge, MA 02139, USA
edemaine@mit.edu

Sarah Eisenstat
MIT Computer Science and Artificial Intelligence Laboratory,
32 Vassar St., Cambridge, MA 02139, USA

Mikhail Rudoy[1]
MIT Computer Science and Artificial Intelligence Laboratory,
32 Vassar St., Cambridge, MA 02139, USA
mrudoy@gmail.com

Abstract

In this paper, we prove that optimally solving an $n \times n \times n$ Rubik's Cube is NP-complete by reducing from the Hamiltonian Cycle problem in square grid graphs. This improves the previous result that optimally solving an $n \times n \times n$ Rubik's Cube with missing stickers is NP-complete. We prove this result first for the simpler case of the Rubik's Square – an $n \times n \times 1$ generalization of the Rubik's Cube – and then proceed with a similar but more complicated proof for the Rubik's Cube case. Our results hold both when the goal is make the sides monochromatic and when the goal is to put each sticker into a specific location.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Combinatorial Puzzles, NP-hardness, Group Theory, Hamiltonicity

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.24

Related Version A full version of the paper is available [4], https://arXiv.org/abs/1706.06708.

1 Introduction

The Rubik's Cube is an iconic puzzle in which the goal is to rearrange the stickers on the outside of a $3 \times 3 \times 3$ cube so as to make each face monochromatic by rotating $1 \times 3 \times 3$ (or $3 \times 1 \times 3$ or $3 \times 3 \times 1$) slices. In some versions where the faces show pictures instead of colors, the goal is to put each sticker into a specific location. The $3 \times 3 \times 3$ Rubik's Cube can be generalized to an $n \times n \times n$ cube in which a single move is a rotation of a $1 \times n \times n$ slice. We can also consider the generalization to an $n \times n \times 1$ figure. In this simpler puzzle, called the $n \times n$ Rubik's Square, the allowed moves are flips of $n \times 1 \times 1$ rows or $1 \times n \times 1$ columns. These two generalizations were introduced in [3].

The overall purpose of this paper is to address the computational difficulty of optimally solving these puzzles. In particular, consider the decision problem which asks for a given

[1] Now at Google.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 24; pp. 24:1–24:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

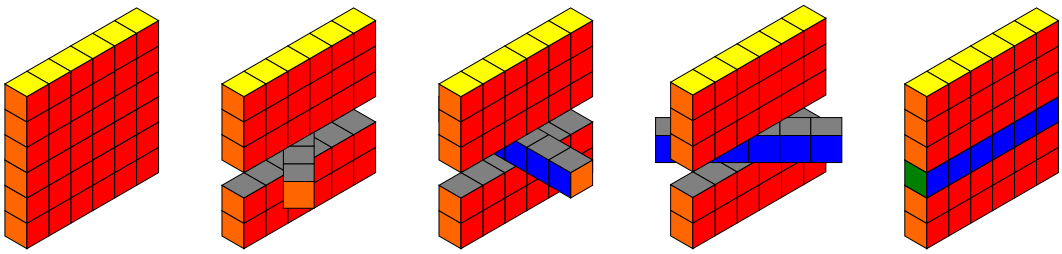


Figure 1 A single move in an example 6×6 Rubik's Square.

puzzle configuration whether that puzzle can be solved in a given number of moves. We show that this problem is NP-complete for the $n \times n$ Rubik's Square and for the $n \times n \times n$ Rubik's Cube under two different move models. These results close a problem that has been repeatedly posed as far back as 1984 [1, 8, 5] and has until now remained open [7].

In Section 2, we formally introduce the decision problems regarding Rubik's Squares and Rubik's Cubes whose complexity we will analyze. Then in Section 3, we introduce the variant of the Hamiltonicity problem that we will reduce from – Promise Cubical Hamiltonian Path – and prove this problem to be NP-hard. Next, we prove that the problems regarding the Rubik's Square are NP-complete in Section 4 by reducing from Promise Cubical Hamiltonian Path. After that, we apply the same ideas in Section 5 to a more complicated proof of NP-hardness for the problems regarding the Rubik's Cube. Finally, we discuss possible next steps in Section 6. Membership in NP, as well as other omitted proofs, can be found in the full version of this paper [4].

2 Rubik's Cube and Rubik's Square problems

2.1 Rubik's Square

We begin with a simpler model based on the Rubik's Cube which we will refer to as the Rubik's Square. In this model, a puzzle consists of an $n \times n$ array of unit cubes, called *cubies* to avoid ambiguity. Every cubie face on the outside of the puzzle has a colored (red, blue, green, white, yellow, or orange) sticker. The goal of the puzzle is to use a sequence of moves to rearrange the cubies such that each face of the puzzle is monochromatic in a different color. A *move* consists of flipping a single row or column in the array through space via a rotation in the long direction as demonstrated in Figure 1.

We are concerned with the following decision problem:

▶ **Problem 1.** *The* ***Rubik's Square*** *problem has as input an $n \times n$ Rubik's Square configuration and a value k. The goal is to decide whether a Rubik's Square in configuration C can be solved in k moves or fewer.*

Note that this type of puzzle was previously introduced in [3] as the $n \times n \times 1$ Rubik's Cube. In that paper, the authors showed that deciding whether it is possible to solve the $n \times n \times 1$ Rubik's Cube in a given number of moves is NP-complete when the puzzle is allowed to have missing stickers (and the puzzle is considered solved if each face contains stickers of only one color).

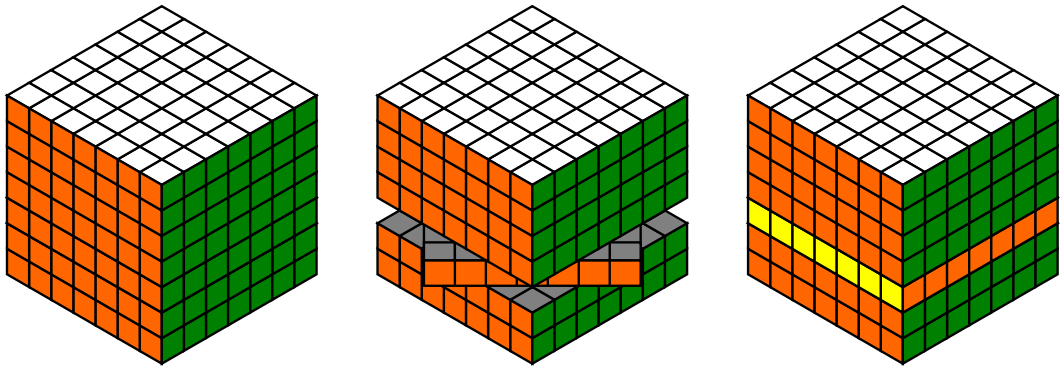


Figure 2 A single slice rotation in an example $7 \times 7 \times 7$ Rubik's Cube.

2.2 Rubik's Cube

Next consider the Rubik's Cube puzzle. An $n \times n \times n$ Rubik's Cube is a cube consisting of n^3 unit cubes called *cubies*. Every face of a cubie that is on the exterior of the cube has a colored (red, blue, green, white, yellow, or orange) sticker. The goal of the puzzle is to use a sequence of moves to reconfigure the cubies in such a way that each face of the cube ends up monochromatic in a different color. A *move count metric* is a convention for counting moves in a Rubik's Cube. Several common move count metrics for Rubik's Cubes are listed in [9]. As discussed in [2], however, many common move count metrics do not easily generalize to $n > 3$ or are not of any theoretical interest. In this paper, we will restrict our attention to two move count metrics called the Slice Turn Metric and the Slice Quarter Turn Metric. Both of these metrics use the same type of motion to define a move. Consider the subdivision of the Rubik's Cube's volume into n *slices* of dimension $1 \times n \times n$ (or $n \times 1 \times n$ or $n \times n \times 1$). In the Slice Turn Metric (STM), a *move* is a rotation of a single slice by any multiple of 90°. Similarly, in the Slice Quarter Turn Metric (SQTM), a *move* is a rotation of a single slice by an angle of 90° in either direction. An example SQTM move is shown in Figure 2.

We are concerned with the following decision problems:

▶ **Problem 2.** *The **STM/SQTM Rubik's Cube problem** takes as input a configuration of a Rubik's Cube together with a number k. The goal is to decide whether a Rubik's Cube in configuration C can be solved in at most k STM/SQTM moves.*

2.3 Notation

Next we define some notation for dealing with the Rubik's Cube and Rubik's Square problems.

To begin, we need a way to refer to cubies and stickers. For this purpose, we orient the puzzle to be axis-aligned. In the case of the Rubik's Square we arrange the $n \times n$ array of cubies in the x and y directions and we refer to a cubie by stating its x and y coordinates. In the case of the Rubik's Cube, we refer to a cubie by stating its x, y, and z coordinates. To refer to a sticker in either puzzle, we need only specify the face on which that sticker resides (e.g. "top" or "$+z$") and also the two coordinates of the sticker along the surface of the face (e.g. the x and y coordinates for a sticker on the $+z$ face).

If $n = 2a + 1$ is odd, then we will let the coordinates of the cubies in each direction range over the set $\{-a, -(a-1), \ldots, -1, 0, 1, \ldots, a-1, a\}$. This is equivalent to centering the puzzle at the origin. If, however, $n = 2a$ is even, then we let the coordinates of the cubies in each direction range over the set $\{-a, -(a-1), \ldots, -1\} \cup \{1, \ldots, a-1, a\}$. In this case, the

coordinate scheme does not correspond with a standard coordinate sheme no matter how we translate the cube. This coordinate scheme is a good idea for the following reason: under this scheme, if a move relocates a sticker, the coordinates of that sticker remain the same up to permutation and negation.

Next, we need a way to distinguish the sets of cubies affected by a move from each other.

In the Rubik's Square, there are two types of moves. The first type of move, which we will call a *row move* or a y *move*, affects all the cubies with some particular y coordinate. The second type of move, which we will call a *column move* or an x *move* affects all the cubies with some particular x coordinate. We will refer to the set of cubies affected by a row move as a *row* and refer to the set of cubies affected by a column move as a *column*. In order to identify a move, we must identify which row or column is being flipped, by specifying whether the move is a row or column move as well as the index of the coordinate shared by all the moved cubies (e.g. the index -5 row move is the move that affects the cubies with $y = -5$).

In the Rubik's Cube, each STM/SQTM move affects a single slice of n^2 cubies sharing some coordinate. If the cubies share an x (or y or z) coordinate, then we call the slice an x *(or y or z) slice*. As with the Rubik's Square, we identify the slice by its normal direction together with its cubies' index in that direction (e.g. the $x = 3$ slice). We will also refer to the six slices at the boundaries of the Cube as *face slices* (e.g. the $+x$ face slice).

A move in a Rubik's Cube can be named by identifying the slice being rotated and the amount of rotation. We split this up into the following five pieces of information: the normal direction to the slice, the sign of the index of the slice, the absolute value of the index of the slice, the amount of rotation, and the direction of rotation. Splitting the information up in this way allows us not only to refer to individual moves (by specifying all five pieces of information) but also to refer to interesting sets of moves (by omitting one or more of the pieces of information).

To identify the normal direction to a slice, we simply specify x, y, or z; for example, we could refer to a move as an x move whenever the rotating slice is normal to the x direction. We will use two methods to identify the sign of the index of a moved slice. Sometimes we will refer to positive moves or negative moves, and sometimes we will combine this information with the normal direction and specify that the move is a $+x$, $-x$, $+y$, $-y$, $+z$, or $-z$ move. We use the term *index-v move* to refer to a move rotating a slice whose index has absolute value v. In the particular case that the slice rotated is a face slice, we instead use the term *face move*. We refer to a move as a *turn* if the angle of rotation is 90° and as a *flip* if the angle of rotation is 180°. In the case that the angle of rotation is 90°, we can specify further by using the terms *clockwise turn* and *counterclockwise turn*. We make the notational convention that clockwise and counterclockwise rotations around the x, y, or z axes are labeled according to the direction of rotation when looking from the direction of positive x, y, or z.

We also extend the same naming conventions to the Rubik's Square moves. For example, a positive row move is any row move with positive index and an index-v move is any move with index $\pm v$.

2.4 Group-theoretic approach

An alternative way to look at the Rubik's Square and Rubik's Cube problems is through the lens of group theory. The transformations that can be applied to a Rubik's Square or Rubik's Cube by a sequence of moves form a group with composition as the group operation. Define RS_n to be the group of possible sticker permutations in an $n \times n$ Rubik's Square and

define RC_n to be the group of possible sticker permutations in an $n \times n \times n$ Rubik's Cube.

Consider the moves possible in an $n \times n$ Rubik's Square or an $n \times n \times n$ Rubik's Cube. Each such move has a corresponding element in group RS_n or RC_n.

For the Rubik's Square, let $x_i \in RS_n$ be the transformation of flipping the column with index i in an $n \times n$ Rubik's Square and let y_i be the transformation of flipping the row with index i in the Square. Then if I is the set of row/column indices in an $n \times n$ Rubik's Square we have that RS_n is generated by the set of group elements $\bigcup_{i \in I} \{x_i, y_i\}$.

Similarly, for the Rubik's Cube, let x_i, y_i, and z_i in RC_n be the transformations corresponding to clockwise turns of x, y, or z slices with index i. Then if I is the set of slice indices in an $n \times n \times n$ Rubik's Cube we have that RC_n is generated by the set of group elements $\bigcup_{i \in I} \{x_i, y_i, z_i\}$.

Using these groups we obtain a new way of identifying puzzle configurations. Let C_0 be a canonical solved configuration of a Rubik's Square or Rubik's Cube puzzle. For the $n \times n$ Rubik's Square, define C_0 to have top face red, bottom face blue, and the other four faces green, orange, yellow, and white in some fixed order. For the $n \times n \times n$ Rubik's Cube, let C_0 have the following face colors: the $+x$ face is orange, the $-x$ face is red, the $+y$ face is green, the $-y$ face is yellow, the $+z$ face is white, and the $-z$ face is blue. Then from any element of RS_n or RC_n, we can construct a configuration of the corresponding puzzle by applying that element to C_0. In other words, every transformation $t \in RS_n$ or $t \in RC_n$ corresponds with the configuration $C_t = t(C_0)$ of the $n \times n$ Rubik's Square or $n \times n \times n$ Rubik's Cube that is obtained by applying t to C_0.

Using this idea, we define a new series of problems:

▶ **Problem 3.** *The **Group Rubik's Square** problem has as input a transformation $t \in RS_n$ and a value k. The goal is to decide whether the transformation t can be reversed by a sequence of at most k transformations corresponding to Rubik's Square moves. In other words, the answer is "yes" if and only if the transformation t can be reversed by a sequence of at most k transformations of the form x_i or y_i.*

▶ **Problem 4.** *The **Group STM/SQTM Rubik's Cube** problem has as input a transformation $t \in RC_n$ and a value k. The goal is to decide whether the transformation t can be reversed by a sequence of at most k transformations corresponding with legal Rubik's Cube moves under move count metric STM/SQTM.*

We can interpret these problems as variants of the Rubik's Square or Rubik's Cube problems. For example, the Rubik's Square problem asks whether it is possible (in a given number of moves) to unscramble a Rubik's Square configuration so that each face ends up monochromatic, while the Group Rubik's Square problem asks whether it is possible (in a given number of moves) to unscramble a Rubik's Square configuration so that each sticker goes back to its exact position in the originally solved configuration C_0. As you see, the Group Rubik's Square problem, as a puzzle, is just a more difficult variant of the puzzle: instead of asking the player to move all the stickers of the same color to the same face, this variant asks the player to move each stickers to the exact correct position. Similarly, the Group STM/SQTM Rubik's Cube problem as a puzzle asks the player to move each sticker to an exact position. These problems can have practical applications with physical puzzles. For example, some Rubik's Cubes have pictures split up over the stickers of each face instead of just monochromatic colors on the stickers. For these puzzles, as long as no two stickers are the same, the Group STM/SQTM Rubik's Cube problem is more applicable than the STM/SQTM Rubik's Cube problem (which can leave a face "monochromatic" but scrambled in image).

We formalize the idea that the Group version of the puzzle is a strictly more difficult puzzle in the following lemmas:

▶ **Lemma 2.1.** *If (t, k) is a "yes" instance to the Group Rubik's Square problem, then $(t(C_0), k)$ is a "yes" instance to the Rubik's Square problem.*

▶ **Lemma 2.2.** *If (t, k) is a "yes" instance to the Group STM/SQTM Rubik's Cube problem, then $(t(C_0), k)$ is a "yes" instance to the STM/SQTM Rubik's Cube problem.*

At this point it is also worth mentioning that the Rubik's Square with SQTM move model is a strictly more difficult puzzle than the Rubik's Square with STM move model:

▶ **Lemma 2.3.** *If (C, k) is a "yes" instance to the SQTM Rubik's Cube problem, then it is also a "yes" instance to the STM Rubik's Cube problem. Similarly, if (t, k) is a "yes" instance to the Group SQTM Rubik's Cube problem, then it is also a "yes" instance to the Group STM Rubik's Cube problem.*

3 Hamiltonicity variants

To prove the problems introduced above hard, we need to introduce several variants of the Hamiltonian cycle and path problems.

It is shown in [6] that the following problem is NP-complete.

▶ **Problem 5.** *A* square grid graph *is a finite induced subgraph of the infinite square lattice. The* Grid Graph Hamiltonian Cycle *problem asks whether a given square grid graph with no degree-*1 *vertices has a Hamiltonian cycle.*

Starting with this problem, we prove that the following promise version of the grid graph Hamiltonian path problem is also NP-hard.

▶ **Problem 6.** *The* Promise Grid Graph Hamiltonian Path *problem takes as input a square grid graph G and two specified vertices s and t with the promise that any Hamiltonian path in G has s and t as its start and end respectively. The problem asks whether there exists a Hamiltonian path in G.*

The above problem is more useful, but it is still inconvenient in some ways. In particular, there is no conceptually simple way to connect a grid graph to a Rubik's Square or Rubik's Cube puzzle. It is the case, however, that every grid graph is actually a type of graph called a "cubical graph". Cubical graphs, unlike grid graphs, can be conceptually related to Rubik's Cubes and Rubik's Squares with little trouble.

So what is a cubical graph? Let H_m be the m dimensional hypercube graph; in particular, the vertices of H_m are the bitstrings of length m and the edges connect pairs of bitstrings whose Hamming distance is exactly one. Then a *cubical graph* is any induced subgraph of any hypercube graph H_m.

Notably, when embedding a grid graph into a hypercube, it is always possible to assign the bitstring label $00\ldots0$ to any vertex. Suppose we start with Promise Grid Graph Hamiltonian Path problem instance (G, s, t); then by embedding G into a hypercube graph, we can reinterpret this instance as an instance of the promise version of cubical Hamiltonian path:

▶ **Problem 7.** *The* Promise Cubical Hamiltonian Path *problem takes as input a cubical graph whose vertices are length-m bitstrings $l_1, l_2, \ldots, l_n$ with the promise that (1) $l_n = 00\ldots0$ and (2) any Hamiltonian path in the graph has l_1 and l_n as its start and end respectively. The*

problem asks whether there exists a Hamiltonian path in the cubical graph. In other words, the problem asks whether it is possible to rearrange bitstrings $l_1, \ldots, l_n$ into a new order such that each bitstring has Hamming distance one from the next.

First, we reduce from the Grid Graph Hamiltonian Cycle problem to the Promise Grid Graph Hamiltonian Path problem.

▶ **Lemma 3.1.** *The Promise Grid Graph Hamiltonian Path problem (Problem 6) is NP-hard.*

Second, we reduce from the Promise Grid Graph Hamiltonian Path problem to the Promise Cubical Hamiltonian Path problem.

▶ **Theorem 3.2.** *The Promise Cubical Hamiltonian Path problem (Problem 7) is NP-hard.*

4 (Group) Rubik's Square is NP-complete

4.1 Reductions

To prove that the Rubik's Square and Group Rubik's Square problems are NP-complete, we reduce from the Promise Cubical Hamiltonian Path problem of Section 3.

Suppose we are given an instance of the Promise Cubical Hamiltonian Path problem consisting of n bitstrings $l_1, \ldots, l_n$ of length m (with $l_n = 00\ldots0$). To construct a Group Rubik's Square instance we need to compute the value k indicating the allowed number of moves and construct the transformation $t \in RS_s$.

The value k can be computed directly as $k = 2n - 1$.

The transformation t will be an element of group RS_s where $s = 2(\max(m, n) + 2n)$. Define a_i for $1 \le i \le n$ to be $(x_1)^{(l_i)_1} \circ (x_2)^{(l_i)_2} \circ \cdots \circ (x_m)^{(l_i)_m}$ where $(l_i)_1, (l_i)_2, \ldots, (l_i)_m$ are the bits of l_i. Also define $b_i = (a_i)^{-1} \circ y_i \circ a_i$ for $1 \le i \le n$. Then we define t to be $a_1 \circ b_1 \circ b_2 \circ \cdots \circ b_n$.

Outputting (t, k) completes the reduction from the Promise Cubical Hamiltonian Path problem to the Group Rubik's Square problem. To reduce from the Promise Cubical Hamiltonian Path problem to the Rubik's Square problem we simply output $(C_t, k) = (t(C_0), k)$. These reductions clearly run in polynomial time.

4.2 Intuition

The key idea that makes this reduction work is that the transformations b_i for $i \in \{1, \ldots, n\}$ all commute. This allows us to rewrite $t = a_1 \circ b_1 \circ b_2 \circ \cdots \circ b_n$ with the b_is in a different order. If the order we choose happens to correspond to a Hamiltonian path in the cubical graph specified by $l_1, \ldots, l_n$, then when we explicitly write the b_is and a_1 in terms of x_js and y_is, most of the terms cancel. In particular, the number of remaining terms will be exactly k. Since we can write t as a combination of exactly k x_js and y_is, we can invert t using at most k x_js and y_is. In other words, if there is a Hamiltonian path in the cubical graph specified by $l_1, \ldots, l_n$, then (t, k) is a "yes" instance to the Group Rubik's Square problem.

In order to more precisely describe the cancellation of terms in t, we can consider just one local part: $b_i \circ b_{i'}$. We can rewrite this as $(a_i)^{-1} \circ y_i \circ a_i \circ (a_{i'})^{-1} \circ y_{i'} \circ a_{i'}$. The interesting part is that $a_i \circ (a_{i'})^{-1}$ will cancel to become just one x_j. Note that

$$a_i \circ (a_{i'})^{-1} = (x_1)^{(l_i)_1} \circ (x_2)^{(l_i)_2} \circ \cdots \circ (x_m)^{(l_i)_m} \circ (x_1)^{-(l_{i'})_1} \circ (x_2)^{-(l_{i'})_2} \circ \cdots \circ (x_m)^{-(l_{i'})_m},$$

which we can rearrange as

$$(x_1)^{(l_i)_1 - (l_{i'})_1} \circ (x_2)^{(l_i)_2 - (l_{i'})_2} \circ \cdots \circ (x_m)^{(l_i)_m - (l_{i'})_m}.$$

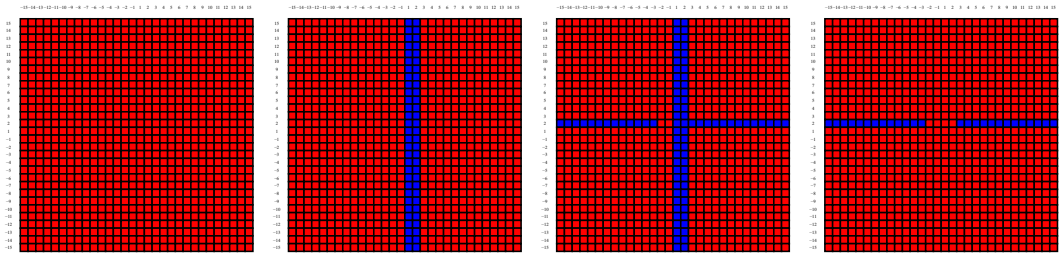


Figure 3 Applying b_2 to C_0 step by step (only top face shown).

Next, if b_i and $b_{i'}$ correspond to adjacent vertices l_i and $l_{i'}$, then $(l_i)_j - (l_{i'})_j$ is zero for all j except one for which $(l_i)_j - (l_{i'})_j = \pm 1$. Thus the above can be rewritten as $(x_j)^1$ or $(x_j)^{-1}$ for some specific j. Since $x_j = (x_j)^{-1}$ this shows that $(a_{i_1})^{-1} \circ a_{i_2}$ simplifies to x_j for some j.

This intuition is formalized in the following sequence of results.

▶ **Lemma 4.1.** *The transformations b_i all commute.*

▶ **Theorem 4.2.** *If $l_1, \ldots, l_n$ is a "yes" instance to the Promise Cubical Hamiltonian Path problem, then (t, k) is a "yes" instance to the Group Rubik's Square problem.*

▶ **Corollary 4.3.** *If $l_1, \ldots, l_n$ is a "yes" instance to the Promise Cubical Hamiltonian Path problem, then (C_t, k) is a "yes" instance to the Rubik's Square problem.*

4.3 Coloring of C_t

In order to show the other direction of the proof, it will be helpful to consider the coloring of the stickers on the top and bottom faces of the Rubik's Square. In particular, if we define $b = b_1 \circ \cdots \circ b_n$ (so that $t = a_1 \circ b$), then it will be very helpful for us to know the colors of the top and bottom stickers in configuration $C_b = b(C_0)$.

Consider for example the instance of Promise Cubical Hamiltonian Path with $n = 5$ and $m = 3$ defined by $l_1 = 011$, $l_2 = 110$, $l_3 = 111$, $l_4 = 100$, $l_5 = 000$. For this example, C_0 is an $s \times s$ Rubik's Square with $s = 2(\max(m, n) + 2n) = 30$.

To describe configuration C_b, we need to know the effect of transformation b_i. For example, Figure 3 shows the top face of a Rubik's Square in configurations C_0, $a_2(C_0)$, $(y_2 \circ a_2)(C_0)$, and $b_2(C_0) = ((a_2)^{-1} \circ y_2 \circ a_2)(C_0)$ where a_2 and y_2 are defined in terms of $l_2 = 110$ as in the reduction.

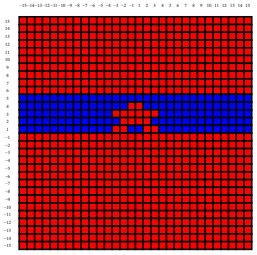
The exact behavior of a Rubik's Square due to b_i is described by the following lemma:

▶ **Lemma 4.4.** *Suppose $i \in \{1, \ldots, n\}$, and $c, r \in \{1, \ldots, s/2\}$. Then*

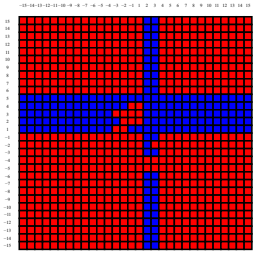
1. *if $r = i$ and $c \leq m$ such that bit c of l_i is 1, then b_i swaps the cubies in positions $(c, -r)$ and $(-c, r)$ without flipping either;*
2. *if $r = i$ and either $c > m$ or $c \leq m$ and bit c of l_i is 0, then b_i swaps the cubies in positions (c, r) and $(-c, r)$ and flips them both;*
3. *all other cubies are not moved by b_i.*

We can apply the above to figure out the effect of transformation $b_1 \circ b_2 \circ \cdots \circ b_n$ on configuration C_0. In particular, that allows us to learn the coloring of configuration C_b.

▶ **Theorem 4.5.** *In C_b, a cubie has top face blue if and only if it is in position (c, r) such that $1 \leq r \leq n$ and either $|c| > m$ or $|c| \leq m$ and bit $|c|$ of l_r is 0.*



(a) The top face of C_b for the example input $l_1, \ldots, l_n$.



(b) The top face of C_t for the example input $l_1, \ldots, l_n$.

Figure 4 The coloring of the Rubik's Square for the example input $l_1, \ldots, l_n$.

This concludes the description of C_b in terms of colors. The coloring of configuration C_t – the configuration that is actually obtained by applying the reduction to $l_1, \ldots, l_n$ – can be obtained from the coloring of configuration C_b by applying transformation a_1.

Applying Theorem 4.5 to the previously given example, we obtain the coloring of the Rubik's Square in configuration C_b as shown in Figure 4a. Note that the $n \times m$ grid of bits comprising $l_1, \ldots, l_n$ is actually directly encoded in the coloring of a section of the Rubik's Square. In addition, the coloring of the Rubik's Square in configuration C_t is shown for the same example in Figure 4b.

4.4 (Group) Rubik's Square solution → Promise Cubical Hamiltonian Path solution

In the full paper, [4], we prove the following theorem:

▶ **Theorem 4.6.** *If (C_t, k) is a "yes" instance to the Rubik's Square problem, then $l_1, \ldots, l_n$ is a "yes" instance to the Promise Cubical Hamiltonian Path problem.*

By Lemma 2.1, this immediately implies the following corollary:

▶ **Corollary 4.7.** *If (t, k) is a "yes" instance to the Group Rubik's Square problem, then $l_1, \ldots, l_n$ is a "yes" instance to the Promise Cubical Hamiltonian Path problem.*

4.5 Conclusion

Theorems 4.2 and 4.6 and Corollaries 4.3 and 4.7 show that the polynomial-time reductions given are answer preserving. As a result, we conclude that

▶ **Theorem 4.8.** *The Rubik's Square and Group Rubik's Square problems are NP-complete.*

5 (Group) STM/SQTM Rubik's Cube is NP-complete

5.1 Reductions

Below, we introduce the reductions used for the Rubik's Cube case. These reductions very closely mirror the Rubik's Square case, and the intuition remains exactly the same: the b_i terms commute, and so if the input Promise Cubical Hamiltonian Path instance is a "yes" instance then the b_is can be reordered so that all but k moves in the definition of t will cancel; therefore in that case t can be both enacted and reversed in k moves.

There are, however, several notable differences from the Rubik's Square case. The first difference is that in a Rubik's Cube, the moves x_i, y_i, and z_i are all quarter turn rotations rather than self-inverting row or column flips. One consequence is that unlike in the Rubik's Square case, the term a_i does not have the property that $(a_i)^{-1} = a_i$. A second difference is that in a Rubik's Square, the rows never become columns or visa versa. In a Rubik's Cube on the other hand, rotation of the faces can put rows of stickers that were once aligned parallel to one axis into alignment with another axis. To avoid allowing a solution of the puzzle due to this fact in the absence of a solution to the input Promise Cubical Hamiltonian Path instance, the slices in this construction which take the role of rows 1 through n in the Rubik's Square case and the slices which take the role of columns 1 through m in the Rubik's Square case will be assigned entirely distinct indices.

To prove that the STM/SQTM Rubik's Cube and Group STM/SQTM Rubik's Cube problems are NP-complete, we reduce from the Promise Cubical Hamiltonian Path problem of Section 3 as described below.

Suppose we are given an instance of the Promise Cubical Hamiltonian Path problem consisting of n biststrings $l_1, \ldots, l_n$ of length m (with $l_n = 00\ldots0$). To construct a Group STM/SQTM Rubik's Square instance we need to compute the value k indicating the allowed number of moves and construct the transformation t in RC_s.

The value k can be computed directly as $k = 2n - 1$.

The transformation t will be an element of group RC_s where $s = 6n + 2m$. Define a_i for $1 \leq i \leq n$ to be $(x_1)^{(l_i)_1} \circ (x_2)^{(l_i)_2} \circ \cdots \circ (x_m)^{(l_i)_m}$ where $(l_i)_1, (l_i)_2, \ldots, (l_i)_m$ are the bits of l_i. Also define $b_i = (a_i)^{-1} \circ z_{m+i} \circ a_i$ for $1 \leq i \leq n$. Then we define t to be $a_1 \circ b_1 \circ b_2 \circ \cdots \circ b_n$.

Outputting (t, k) completes the reduction from the Promise Cubical Hamiltonian Path problem to the Group STM/SQTM Rubik's Cube problem. To reduce from the Promise Cubical Hamiltonian Path problem to the STM/SQTM Rubik's Cube problem we simply output $(C_t, k) = (t(C_0), k)$. As with the Rubik's Square case, these reductions are clearly polynomial-time reductions.

5.2 Promise Cubical Hamiltonian Path solution → (Group) STM/SQTM Rubik's Cube solution

The proof of this direction is not substantively different from the proof of the first direction for the Rubik's Square problems. The differences in these proofs are all minor details that are only present to account for the differences (listed above) between the Rubik's Square and Rubik's Cube reductions. See [4], the full paper, for details.

5.3 Coloring of C_t

As in the Rubik's Square case, it is helpful for the second direction of the proof to know the coloring of the Cube's configuration. As before, we define $b = b_1 \circ \cdots \circ b_n$ (so that $t = a_1 \circ b$) and determine the colors of the stickers in configuration $C_b = b(C_0)$.

Consider the example instance of Promise Cubical Hamiltonian Path with $n = 5$ and $m = 3$ introduced in Section 4.3. For this example instance, the Rubik's Cube configuration produced by the reduction is an $s \times s \times s$ Rubik's Cube with $s = 2m + 6n = 36$. Furthermore, the coloring of the stickers in C_b for this example is shown in Figure 5. Note that the $n \times m$ grid of bits comprising $l_1, \ldots, l_n$ is actually directly encoded in the coloring of each face.

In the full paper, we formalize the general pattern of colors intuited from this example.

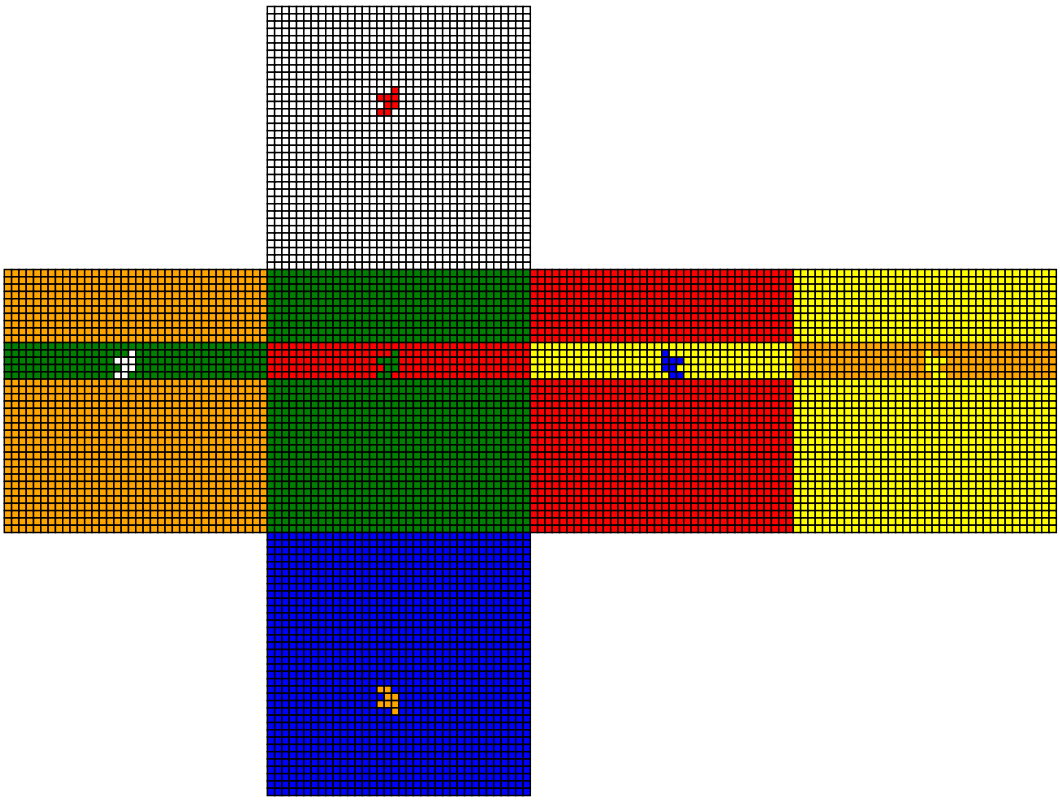


Figure 5 The faces of C_b for the example input $l_1, \ldots, l_n$. In this figure, the top and bottom faces are the $+z$ and $-z$ faces, while the faces in the vertical center of the figure are the $+x$, $+y$, $-x$, and $-y$ faces from left to right.

5.4 (Group) STM/SQTM Rubik's Cube solution → Promise Cubical Hamiltonian Path solution: proof outline

In [4], the full paper, we prove the following:

▶ **Theorem 5.1.** *If (C_t, k) is a "yes" instance to the STM Rubik's Cube problem, then $l_1, \ldots, l_n$ is a "yes" instance to the Promise Cubical Hamiltonian Path problem.*

By Lemmas 2.2 and 2.3, this immediately implies the following corollary:

▶ **Corollary 5.2.** *If (t, k) is a "yes" instance to the Group STM/SQTM Rubik's Cube problem or (C_t, k) is a "yes" instance to the STM/SQTM Rubik's Cube problem, then $l_1, \ldots, l_n$ is a "yes" instance to the Promise Cubical Hamiltonian Path problem.*

The intuition behind the proof of Theorem 5.1 is similar to that used in the Rubik's Square case, but there is added complexity due to the extra options available in a Rubik's Cube. Most of the added complexity is due to the possibility of face moves (allowing rows of stickers to align in several directions over the course of a solution). Below, we describe an outline of the proof using several high-level steps.

Consider a hypothetical solution to the (C_t, k) instance of the STM Rubik's Cube problem consisting of a sequence of STM Rubik's Cube moves $m_1, \ldots, m_{k'}$ with $k' \leq k$ such that $C' = (m_{k'} \circ \cdots \circ m_1)(C_t)$ is a solved configuration of the Rubik's Cube.

Using the fact that the side-length of the cube is large compared to the number of allowed moves, we prove the following preliminary facts:

- There are no indices $i \in \{1, \ldots, n\}$ such that $m_1, \ldots, m_{k'}$ contains exactly zero index-$(m+i)$ moves.
- If $m_1, \ldots, m_{k'}$ contains exactly one index-$(m+i)$ move, then the sole index-$(m+i)$ move must be a counterclockwise z turn. In this case, call the move in question an O-move (where O stands for "one").
- If $m_1, \ldots, m_{k'}$ contains exactly two index-$(m+i)$ moves, then the two index-$(m+i)$ moves must be a clockwise z turn and a z flip in some order. In this case, call the moves in question T-moves (where T stands for "two").
- All O- and T-moves must occur at a time when faces $+x$, $+y$, $-x$, and $-y$ all have zero rotation and any move of z slice $-(m+i)$ must occur at a time when these faces all have rotation 180°.

Next, we introduce a new concept, paired stickers, and use it to prove the following. Suppose that $j \in \{1, 2, \ldots, m\}$ is a value such that l_{i_1} and l_{i_2} differ in bit j, and $i_1, i_2 \in \{1, \ldots, n\}$ are indices for which $m_1, \ldots, m_{k'}$ contains an index-$(m+i_1)$ O-move and an index-$(m+i_2)$ O-move. Then it must be the case that between these two moves there is either at least one index-j move or at least one face move of faces $+x$, $+y$, $-x$, and $-y$ (which by the previous results actually requires at least two such face moves).

After that, we use a counting argument to significantly restrict the possible moves in $m_1, \ldots, m_{k'}$. In particular, we classify the moves into several types and use the previous results to bound the number of moves of each type. Adding these bounds together and simplifying, we find that $k' \geq k$. Since we already know that $k' \leq k$, we learn that equality must hold in each of our computed bounds. This immediately constrains the quantity of each type of move even further. In particular, we learn that $m_1, \ldots, m_{k'}$ consists of three types of moves: O-moves, T-moves, and index-j moves for $j \in \{1, 2, \ldots, m\}$ (call these J-moves). Furthermore, there is exactly one J-move between every consecutive pair of O-moves.

After the types of moves in $m_1, \ldots, m_{k'}$ are restricted to this extent, several possibilites that we previously had to consider are no longer relevent (i.e. there are no face moves). As a consequence, the earlier results are actually strengthened and can be reapplied to learn even more about the types of O-, T-, and J-moves in $m_1, \ldots, m_{k'}$.

Finally, by applying the idea of paired stickers to our now-highly-constrained sequence of moves $m_1, \ldots, m_{k'}$, we are able to show that there are no T-moves in $m_1, \ldots, m_{k'}$. At this point, we can conclude that $m_1, \ldots, m_{k'}$ consists entirely of alternating O- and J-moves with one O-move of an index-$(m+i)$ slice for every $i \in \{1, \ldots, n\}$. If three consecutive O-, J-, and O-moves rotate index-$(m+i_1)$, index-j, and index-$(m+i_2)$ slices, then it must be the case that l_{i_1} and l_{i_2} differ in bit j and in no other bit. Thus, if we consider all of the O-moves in the order in which they occur, the corresponding elements $i \in \{1, \ldots, n\}$ in the same order have the property that each bitstring l_i is at Hamming distance one from the next. In other words, we have our desired result: that $l_1, \ldots, l_n$ is a "yes" instance to the Promise Cubical Hamiltonian Path problem.

5.5 Conclusion

Sections 5.2 and 5.4 show that the polynomial-time reductions given are answer preserving. As a result, we conclude that

▶ **Theorem 5.3.** *The STM/SQTM Rubik's Cube and Group STM/SQTM Rubik's Cube problems are NP-complete.*

6 Future work

In this paper, we resolve the complexity of optimally solving Rubik's Cubes under move count metrics for which a single move rotates a single slice. It could be interesting to consider the complexity of this problem under other move count metrics.

Of particular interest are the Wide Turn Metric (WTM) and Wide Quarter Turn Metric (WQTM), in which the puzzle solver can rotate any contiguous group of layers including a face. These metrics correspond most directly to how one would physically solve a real-world $n \times n \times n$ Rubik's Cube: by grabbing some number of layers (including a face) from the side of the cube and rotating thm together. We can also consider the $1 \times n \times n$ analogue of the Rubik's Cube with WTM move count metric: this would be a Rubik's Square in which a single move flips a contiguous sequence of rows or columns including a row or column at the edge of the Square. Solving this toy model could help point us in the right direction for the WTM and WQTM Rubik's Cube problems. If the toy model resists analysis, it could be interesting to consider this toy model with missing stickers.

References

1 Stephen A. Cook. Can computers routinely discover mathematical proofs? *Proceedings of the American Philosophical Society*, 128(1):40–43, 1984. URL: http://www.jstor.org/stable/986492.

2 Cride5. Move count metrics for big cubes - standards and preferences. Speed Solving Forum, August 2010. URL: https://www.speedsolving.com/forum/showthread.php?23546-Move-count-metrics-for-big-cubes-standards-and-preferences.

3 Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Anna Lubiw, and Andrew Winslow. Algorithms for solving Rubik's Cubes. In *Proceedings of the 19th European Conference on Algorithms*, ESA'11, pages 689–700, Berlin, Heidelberg, 2011. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=2040572.2040647.

4 Erik D. Demaine, Sarah Eisenstat, and Mikhail Rudoy. Solving the Rubik's Cube optimally is NP-complete. arXiv:1706.06708, 2017. URL: https://arXiv.org/abs/1706.06708.

5 Jeff Erickson. Is optimally solving the n×n×n Rubik's Cube NP-hard? Theoretical Computer Science Stack Exchange. URL: https://cstheory.stackexchange.com/q/783 (version: 2010-10-23).

6 Alon Itai, Christos H. Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, November 1982.

7 Graham Kendall, Andrew J. Parkes, and Kristian Spoerer. A survey of NP-complete puzzles. *ICGA Journal*, 31:13–34, 2008.

8 Daniel Ratner and Manfred K. Warmuth. Nxn puzzle and related relocation problem. *J. Symb. Comput.*, 10(2):111–138, 1990. doi:10.1016/S0747-7171(08)80001-6.

9 Wiki. Metric. Speed Solving Wiki, May 2010. URL: https://www.speedsolving.com/wiki/index.php/Metric.

Approximation Algorithms for Scheduling with Resource and Precedence Constraints

Gökalp Demirci
Department of Computer Science, University of Chicago
1100 East 58th Street, Chicago, Illinois 60615, USA
demirci@cs.uchicago.edu

Henry Hoffmann
Department of Computer Science, University of Chicago
1100 East 58th Street, Chicago, Illinois 60615, USA
hankhoffmann@cs.uchicago.edu

David H. K. Kim
Department of Computer Science, University of Chicago
1100 East 58th Street, Chicago, Illinois 60615, USA
hongk@cs.uchicago.edu

Abstract

We study non-preemptive scheduling problems on identical parallel machines and uniformly related machines under both resource constraints and general precedence constraints between jobs. Our first result is an $O(\log n)$-approximation algorithm for the objective of minimizing the makespan on parallel identical machines under resource and general precedence constraints. We then use this result as a subroutine to obtain an $O(\log n)$-approximation algorithm for the more general objective of minimizing the total weighted completion time on parallel identical machines under both constraints. Finally, we present an $O(\log m \log n)$-approximation algorithm for scheduling under these constraints on uniformly related machines. We show that these results can all be generalized to include the case where each job has a release time. This is the first upper bound on the approximability of this class of scheduling problems where both resource and general precedence constraints must be satisfied simultaneously.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Scheduling, Resource, Precedence, Weighted Completion Time

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.25

Related Version The full version of the paper is available as University of Chicago, Department of Computer Science Technical Report with report number TR-2018-01 at `https://newtraell.cs.uchicago.edu/research/publications/techreports/TR-2018-01`.

Funding Funding for this work comes from a DOE Early Career Research Award.

Acknowledgements We thank the anonymous reviewers for their time and feedback. We also thank Janos Simon for his valuable comments on an earlier version of this work.

1 Introduction

Scheduling under resource constraints and scheduling under precedence constraints are both well studied topics in scheduling theory and approximation algorithms. In the former, each job has a resource requirement and there is a finite amount of resource; when jobs run in

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 25; pp. 25:1–25:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

parallel their total resource requirement must not exceed the given resource capacity. In the latter, a precedence relationship between jobs is given; if a job j has precedence over another, say j', then j must be completed before j' can start. Both of these problems are central to scheduling theory, and their approximability is well understood. They have $(2+\epsilon)$- and $(2-1/m)$-approximation algorithms, respectively, where m is the number of identical machines. However, the approximability of the natural problem with the combination of these constraints is still wide open. We study this problem, namely scheduling on parallel machines under both resource and precedence constraints, and give the first non-trivial approximation algorithms for several important versions of this problem.

The combination of these two constraints naturally models the computation in emerging high performance computing systems. Power consumption is one of the central design considerations for the next generation of *exascale* supercomputers [16]. These future systems will have to run parallel computations within a 20 MW operating budget, but will be built such that if all processors were used at full capacity the budget would be exceeded and the system would fail catastrophically [10]. Therefore, exascale system software must schedule parallel computations (with precedence constraints) on this hardware while respecting the global power budget (a limited shared resource) and minimizing application runtime [18]. This critical problem for emerging supercomputers is a practical example of a scheduling with simultaneous resource and precedence constraints.

We now define the problem formally. We are given a set $\mathcal{J}$ of n jobs to be scheduled on m parallel machines. The schedule needs to be non-preemptive; i.e. each job, once assigned to a machine at some point in time, must run to completion on the same machine without interruption. Each job $j \in \mathcal{J}$ has a processing time $p_j \in \mathbb{Z}_{\geq 0}$ and a resource requirement $s_j \in \mathbb{Z}_{\geq 0}$. There is a global resource capacity $S \in \mathbb{Z}_{\geq 0}$ which any feasible schedule must respect. That is, the sum of the resource requirements of the jobs running on the m machines at any point in time must be at most S. Moreover, we have precedence constraints given by a partial order $\prec$ on the jobs such that if $j \prec j'$ then j must complete before j' can start. We study both *identical* machines and *uniformly related* machines. The difference between the two is the time it takes to process a job—on identical machines, it takes p_j units of time to complete job $j \in \mathcal{J}$, regardless of the machine it runs on. In the case of uniformly related machines, we are given as input a speed f_i $(0 < f_i \leq 1)$ associated with each machine i for $1 \leq i \leq m$, and it takes p_j/f_i units of time to complete job j on machine i.

We also consider two different objectives. The first is minimizing the makespan, or the final completion time of all jobs. This problem on identical machines is denoted as $P|\text{res1}, \text{prec}|C_{max}$ in the standard scheduling notation introduced in [6]. Here P denotes identical parallel machines, as opposed to uniformly related machines which is denoted by Q. Also, the resource constraint, res1, indicates that we have a single resource, and prec stands for the general precedence constraint. C_{max} indicates that the objective is to minimize the makespan. The other objective we consider is the more general goal of minimizing the total weighted completion time. In this setting there is a set of weights $\{w_j\}_{j\in\mathcal{J}}$ associated with the jobs, and the goal is to minimize $\sum_{j\in\mathcal{J}} w_j C_j$ where C_j is the completion time of job j in the final schedule. Using the same notation, we denote this problem on identical machines as $P|\text{res1}, \text{prec}|\sum_j w_j C_j$. Corresponding problems on uniformly related machines are denoted as $Q|\text{res1}, \text{prec}|C_{max}$ and $Q|\text{res1}, \text{prec}|\sum_j w_j C_j$, respectively. Note that the minimum total weighted completion time objective $(\sum_j w_j C_j)$ is more general than the minimum makespan objective (C_{max}) in the presence of precedence constraints as one could transform a C_{max} objective into a $\sum_j w_j C_j$ objective by simply setting all w_j's to be 0 and adding a "last" job j' with $p_{j'} = 0$, $s_{j'} = 0$, and $w_{j'} = 1$ which depends on every other job.

In addition, we consider these problems with release time constraints. In this case, we have a release time $r_j \in \mathbb{Z}_{\geq 0}$ associated with each job in the input such that j can only be started after time r_j. We add r_j in the middle constraint section in the scheduling notation to denote the problems with the additional release time constraint (e.g. $P|\text{res1}, \text{prec}, r_j|C_{max}$).

We have obtained, for the first time, approximation results for all these scheduling problems with provable upper bounds. Our most general result is an $O(\log m \log n)$-approximation for weighted completion time on uniformly related machines under resource and precedence constraints with release times (Theorem 12), which makes substantial use of our new results for more restricted models as well as new specialized linear programming schemes.

Related Work

One important class of problems is scheduling subject to only precedence constraints (e.g. $P|\text{prec}|C_{max}$, $Q|\text{prec}|\sum_j w_j C_j$). Almost all variants of these problems have been studied extensively. For the case of identical parallel machines and the minimum makespan objective ($P|\text{prec}|C_{max}$), Graham's seminal list scheduling algorithm [7] gives a $(2-1/m)$-approximation. We will utilize this algorithm and explain some aspects of its analysis in Section 2.1. An almost matching $(2-\epsilon)$-hardness of approximation result is obtained by Svensson [17] assuming a stronger version of the Unique Game Conjecture. For the more general weighted completion time objective ($P|\text{prec}|\sum_j w_j C_j$), Hall et al. [8] gave a 7-approximation which was later improved to a 4-approximation by Munier, Queyranne and Schulz [12]. Very recently, Li [11] obtained the current best ratio of $(2+2\ln 2+\epsilon)$. Of course, all the hardness results for minimizing makespan hold for minimizing the weighted total completion time; however, no stronger hardness results are known for this apparently harder problem. For related machines running at different speeds—$Q|\text{prec}|\sum_j w_j C_j$ and $Q|\text{prec}|C_{max}$—Chudak and Shmoys [3] gave an $O(\log m)$-approximation by grouping the machines into $\log m$ groups according to their speed and treating the machines in each group as identical parallel machines. Li [11] improved this ratio to $O(\log m/\log\log m)$. On the negative side, Bazzi and Norouzi-Fard [2] recently showed the problem is hard to approximate for any constant assuming the hardness of an optimization problem on k-partite graphs.

Another overlapping set of problems is resource-constrained scheduling with one common resource (e.g., $P|\text{res1}|C_{max}$ and $Q|\text{res1}|\sum_j w_j C_j$). Garey and Graham's result in [5] implies a $(3-3/m)$-approximation for $P|\text{res1}|C_{max}$. Later, Niemeier and Wiese [13] gave the algorithm with the current best approximation ratio of $(2+\epsilon)$. Recently, Jansen et al. [9] provided an AFPTAS for the problem. Since bin packing is a special case of this problem, a simple reduction from the partition problem shows that no approximation with a ratio smaller than 3/2 is possible unless $P = NP$. Note that, in all the problems we consider here, preemption is not allowed and the processing time of a job does not depend on the resource allocated to the job.

There is a clear connection between resource constrained scheduling and packing problems, such as bin packing and strip packing. In the case of unit processing times ($p_j = 1$), for example, $P|\text{res1}, p_j = 1|C_{max}$ is the same as the bin packing problem where we pack different sized items into bins of fixed size and try to minimize the number of bins used. In the strip packing problem, we have a fixed width strip with one open end and a finite set of rectangles. The goal is to fit all the rectangles in the strip minimizing the total height that the rectangles reach. The correspondence between strip packing and resource constrained scheduling is obvious. The width of the strip corresponds to the global resource limit and the widths and the heights of the rectangles correspond to the resource requirement and the processing time of the jobs respectively. In fact, we make use of this correspondence between the two

problems in our algorithms. One difference between these two problems is that we can pack as many small width rectangles as the width of the strip allows in the strip packing problem whereas the number of jobs that can run in parallel is bounded by the number of machines in the resource constrained scheduling problem.

Our Results and Techniques

We study identical parallel machines in Section 2. First, in Section 2.1, we consider the objective of minimizing the makespan under both resource and precedence constraints, namely the problem $P|\text{res1}, \text{prec}|C_{max}$. We prove the following.

▶ **Theorem 1.** *There is a* $2 + 2\log(n+1)$*-approximation algorithm for* $P|res1, prec|C_{max}$.

To show this, we consider the two constraints separately and present a two-step algorithm for minimizing the makespan. The first step produces an intermediate approximate schedule satisfying only the precedence constraints with a loss of factor of 2 in the approximation. The second step uses a divide and conquer algorithm (inspired by [1]) to stretch the intermediate schedule so that it also satisfies the resource constraint. We generalize this result for the version of the problem with release times.

▶ **Theorem 7.** *There is a* $2 + 4\log(n+1)$*-approximation algorithm for* $P|res1, prec, r_j|C_{max}$.

In Section 2.2, we use the algorithm from Section 2.1 as a subroutine to obtain our first main result:

▶ **Theorem 8.** *There is an* $O(\log n)$*-approximation algorithm for* $P|res1, prec, r_j|\sum_j w_j C_j$.

To obtain this result, we extend the general framework of [8], [15], and [3] with a novel Linear Programming (LP) relaxation for $P|\text{res1}, \text{prec}, r_j|\sum_j w_j C_j$. In this framework, we divide the time horizon into geometrically increasing intervals. Using our new linear program, we obtain the *approximate completion interval* for each job. Then we show that if we consider, for each interval, the set of jobs completing in the same interval as a separate instance of $P|\text{res1}, \text{prec}, r_j|C_{max}$ problem and schedule them in a separate fragment using the makespan minimizing algorithm we obtain in Section 2.1, the concatenation of these fragments also gives a good approximation to the minimum total weighted completion time objective ($\sum_j w_j C_j$) for the original instance. Our core technique is our time-interval-indexed LP where we carefully incorporate the resource requirements into the LP to guarantee a reasonable total resource requirement by the set of jobs completing in any given interval. This allows us to bound the makespan given by our algorithm from Section 2.1 for each $P|\text{res1}, \text{prec}, r_j|C_{max}$ instance.

Finally, in Section 3, we build on our previous techniques to show that they can be modified to work together with the methods in [3] to get an $O(\log m \log n)$-approximation for scheduling on uniformly related machines subject to simultaneous resource, precedence, and release times constraints with weighted completion time objective. In particular, we state a simple generalization of the time-interval-indexed LP used in the previous section without the resource LP constraints) to machines with different speeds. We use the LP solution to get the approximate completion time intervals as we did for identical machines setting as well as job to machine assignments. Then, we argue that the first step of our two-step makespan minimizing algorithm can be replaced by the $O(\log m)$-approximation algorithm for $Q|\text{prec}, r_j|C_{max}$ given in [3]. Moreover, the second step of our algorithm will respect these job to machine assignments. If we begin with an optimal solution to the LP, we show that these integral job to machine assignments are close enough to the fractional assignments given by the LP solution.

▶ **Theorem 12.** *There is an $O(\log m \log n)$-approximation algorithm for $Q|res1,\ prec,\ r_j|\sum_j w_j C_j$.*

These are the first algorithms with non-trivial approximation ratios for this class of scheduling problems where a resource constraint and general precedence constraints need to be satisfied together.

2 Identical Parallel Machines

2.1 The Minimum Makespan Objective

In this section, we present an $O(\log n)$-approximation algorithm for the problem of minimizing the makespan under both resource and precedence constraints.

▶ **Theorem 1.** *There is a $2 + 2\log(n+1)$-approximation algorithm for $P|res1, prec|C_{max}$.*

Given an instance $(\mathcal{J}, m, \{p_j\}_{j\in\mathcal{J}}, S, \{s_j\}_{j\in\mathcal{J}}, \prec)$ of $P|\text{res1}, \text{prec}|C_{max}$, we let OPT denote the makespan of a minimum makespan schedule of this instance. Our algorithm solves the problem in two steps by handling its precedence and resource constraints separately. First, we consider the corresponding $P|\text{prec}|C_{max}$ instance where we drop the resource requirement (i.e. $(\mathcal{J}, m, \{p_j\}_{j\in\mathcal{J}}, \prec)$). In his seminal work, Graham [7] presents an online machine-driven list scheduling algorithm for this problem. His algorithm greedily considers the jobs in some arbitrary extension of the partial order $\prec$ to a total order (i.e. a list). As soon as a machine is idle, the algorithm schedules the next available job (i.e. all its predecessors are finished) in the list on that machine. In the analysis, he uses two standard lower bounds for the value of OPT:

▶ **Lemma 2** (Load Bound). $\frac{1}{m}\sum_{j\in\mathcal{J}} p_j \le OPT$.

▶ **Lemma 3** (Chain Bound). $\max_{\mathcal{C}\ is\ a\ chain} \sum_{j\in\mathcal{C}} p_j \le OPT$.

The Load Bound is implied by the observation that a perfectly balanced schedule with no idle time would have a makespan of $\frac{1}{m}\sum_{j\in\mathcal{J}} p_j$. Also note that the total processing time of any "chain" of precedence constraints such as $j_1 \prec j_2 \prec \cdots \prec j_k$ is a lower bound on the makespan of any schedule. His analysis charges the time intervals where all the machines are busy on the Load Bound and the time intervals where some machines are idle on the Chain Bound. We run Graham's list scheduling algorithm on our instance after we drop the resource constraints and get an approximate intermediate schedule satisfying the precedence constraints. Let $\mathsf{LS}_{\mathcal{J}}$ denote the makespan of this schedule and $\mathsf{LS}(j)$ denote the completion time of a job $j \in \mathcal{J}$ in it. Graham [7] shows this makespan is bounded by the sum of the Load and the Chain Bounds $\mathsf{LS}_{\mathcal{J}} \le \frac{1}{m}\sum_{j\in\mathcal{J}} p_j + \max_{\mathcal{C}\text{ is a chain}} \sum_{j\in\mathcal{C}} p_j \le 2\cdot OPT$.[1]

The schedule we get by running Graham's list scheduling on the corresponding $P|\text{prec}|C_{max}$ instance may violate the resource constraints of the original instance. In the second step of the algorithm, we run a divide and conquer algorithm on this schedule to make it satisfy the resource constraints without disturbing the precedences already satisfied after our first step. We lose a factor of 2 in the approximation due to Graham's algorithm and an $O(\log n)$-factor in the second step. In the rest of this section, we explain and analyze the second step of the algorithm in detail.

[1] In fact, [7] shows a slightly stronger bound of $(2 - 1/m)\,OPT$.

Algorithm 1 $\mathbf{DS}(J, B, E)$ Divide and Schedule.

Input: A subset of jobs $J \subseteq \mathcal{J}$ and the beginning B and the end E times of J in the schedule of the first step

Output: A makespan y for a feasible schedule of J

1: **if** $J = \emptyset$ **then**
2: **return** 0
3: **end if**
4: $J_{bef} \leftarrow \{j \in J \mid \mathsf{LS}(j) < (B+E)/2\}$
5: $J_{mid} \leftarrow \{j \in J \mid \mathsf{LS}(j) - p_j < (B+E)/2 \text{ and } \mathsf{LS}(j) \geq (B+E)/2\}$
6: $J_{aft} \leftarrow \{j \in J \mid \mathsf{LS}(j) - p_j \geq (B+E)/2\}$
7: $y_{bef} \leftarrow \mathbf{DS}(J_{bef}, \min_{j \in J_{bef}} \mathsf{LS}(j) - p_j, \max_{j \in J_{bef}} \mathsf{LS}(j))$
8: Schedule J_{bef} according to $\mathbf{DS}(J_{bef}, \min_{j \in J_{bef}} \mathsf{LS}(j) - p_j, \max_{j \in J_{bef}} \mathsf{LS}(j))$.
9: $y_{mid} \leftarrow \mathbf{NFDH}(J_{mid})$
10: Schedule J_{mid} according to $\mathbf{NFDH}(J_{mid})$ starting at y_{bef}.
11: $y_{aft} \leftarrow \mathbf{DS}(J_{aft}, \min_{j \in J_{aft}} \mathsf{LS}(j) - p_j, \max_{j \in J_{aft}} \mathsf{LS}(j))$
12: Schedule J_{aft} according to $\mathbf{DS}(J_{aft}, \min_{j \in J_{aft}} \mathsf{LS}(j) - p_j, \max_{j \in J_{aft}} \mathsf{LS}(j))$ starting at $y_{bef} + y_{mid}$.
13: **return** $y_{bef} + y_{mid} + y_{aft}$

Note that the total resource required to complete a job $j \in \mathcal{J}$ is its resource requirement integrated over the time it takes to complete the job: $\int_0^{p_j} s_j \, \mathrm{d}t = s_j p_j$. We denote this value by $\mathsf{RB}(j)$. We define $\mathsf{RB}(J)$ for any subset $J \subseteq \mathcal{J}$ of jobs as the sum of the total resource required for jobs in J (i.e. $\mathsf{RB}(J) = \sum_{j \in J} \mathsf{RB}(j)$). Our algorithm uses another lower bound derived by comparing the resource available in a makespan and total resource required by all jobs.

▶ **Lemma 4** (Resource Bound). $\mathsf{RB}(\mathcal{J})/S \leq OPT$.

To incorporate the resource constraint into the schedule obtained in the first step, we run a divide and conquer algorithm similar to the one employed by Augustine et al. [1] for strip packing. This algorithm (Algorithm 1: **DS**) will need to use a subroutine for scheduling at most m jobs $J \subseteq \mathcal{J}$ with resource constraints that have no precedence constraints among them on m machines. We denote this subroutine by Next-Fit Decreasing-Height: $\mathbf{NFDH}(J)$ and it guarantees a makespan that is bounded by $\mathbf{NFDH}(J) \leq 2\mathsf{RB}(J)/S + \max_{j \in J} p_j$. A simple greedy algorithm that schedules the jobs respecting their resource constraints in the order of non-increasing processing time will have this guarantee. Next-Fit Decreasing-Height algorithm for strip packing analyzed in [4] is one such algorithm when applied to scheduling m jobs on m machines with resource constraints and no precedence constraints.

The algorithm is called with the following initial arguments: all jobs $\mathcal{J}$, beginning time $B = 0$, and end time $E = \mathsf{LS}_{\mathcal{J}}$, the makespan of the schedule obtained in the first step. It takes the set of jobs J_{mid} scheduled to cross the mid time point $\mathsf{LS}_{\mathcal{J}}/2$ in the schedule obtained in the first step. It schedules these jobs in a separate time fragment and concatenates it with schedules obtained recursively on the jobs before, J_{bef}, and the jobs after, J_{aft}. We need the following lemma to justify the initial condition required to call $\mathbf{NFDH}(J_{mid})$. Proofs of all the lemmas in the rest of the paper can be found in the full version of the paper.

▶ **Lemma 5.** *The jobs in J_{mid} have no precedence constraints among them and $1 \leq |J_{mid}| \leq m$.*

In any given level of the recursion tree, the total loss across all recursive calls in this level is at most an additive factor of $O(OPT)$. Since the algorithm terminates in at most $\lceil \log n \rceil$ levels, we get an $O(\log n)$ approximation. Lemma 6 proves this formally and gives a bound on our initial call to Divide and Schedule algorithm: $\mathbf{DS}(\mathcal{J}, 0, \mathsf{LS}_{\mathcal{J}}) \leq 2\mathsf{RB}(\mathcal{J})/S + \mathsf{LS}_{\mathcal{J}} \log(|\mathcal{J}|+1) \leq (2+2\log(n+1))OPT$. This completes the proof of Theorem 1.

▶ **Lemma 6.** *$\boldsymbol{DS}(J, B, E) \leq 2\mathsf{RB}(J)/S + (E-B)\log(|J|+1)$*

The first step of our algorithm can be extended to accommodate release times constraints:

▶ **Theorem 7.** *There is a $2+4\log(n+1)$-approximation algorithm for $P|res1, prec, r_j|C_{max}$.*

In addition to resource and precedence constraints, each job $j \in \mathcal{J}$ now has a release time r_j such that j can only be started after time r_j ($P|\text{res1}, \text{prec}, r_j|C_{max}$). Munier, Queyranne and Schulz [12, 14] provide a 4-approximation for $P|\text{prec}, r_j|C_{max}$. We replace Graham's list scheduling in the first step of our algorithm with their 4-approximation algorithm to get a similar bound for the problem with all three constraints. Note that, after the first step, we obtain a schedule that satisfies the release times and precedence constraints. The second step of our algorithm can easily be made to never schedule a job before its starting time in the schedule from the first step (by forcing J_{mid}'s fragment to start at $\max\{(E-B)/2, y_{bef}\}$).

We note that an $o(\log n)$-approximation is not possible using only the Load, Chain, and Resource Bounds. The bottleneck is in the second step where we use divide-and-conquer. The gap example in [1] for strip packing shows one needs stronger lower bounds on OPT than "the area bound" (Resource Bound in our setting) and "the longest chain of rectangles bound" (Chain Bound in our setting) for an $o(\log n)$ approximation. Their example can easily be translated into a scheduling instance with $m = \log n$ machines. Setting $m = \log n$ allows any solution to their example be interpreted as a scheduling solution because the example has at most $\log n$ parallel precedence chains at any point. Also, the Load Bound of the translated instance is at most $\Theta(1)$, where $OPT = \Omega(\log n)$.[2] Thus, for both the makespan and the weighted completion time objectives, one needs stronger lower bounds on OPT than any combination of the three bounds we use for a better approximation ratio.

2.2 The Minimum Weighted Completion Time Objective

In this section, we generalize our result by giving an $O(\log n)$-approximation algorithm for the minimum total weighted completion time objective. In addition to processing time p_j, resource requirement s_j, and release time r_j, we now have a weight w_j associated with each job $j \in \mathcal{J}$ and our goal is to minimize the total weighted completion time $\sum_j w_j C_j$, where C_j is the completion time of j in the final schedule. This problem is denoted as $P|\text{res1}, \text{prec}, r_j|\sum_j w_j C_j$.

▶ **Theorem 8.** *There is an $O(\log n)$-approximation algorithm for $P|res1, prec, r_j|\sum_j w_j C_j$.*

We first reduce the problem of minimizing weighted completion time ($\sum_j w_j C_j$) to a set of smaller problems with the objective of minimizing the makespan (C_{max}). We then use our $O(\log n)$-approximation algorithm from Section 2.1 for the minimum makespan objective as a subroutine on these problems. In the reduction, we use the general framework of Hall et al.

[2] See [1] and their illustrations for a detailed description of the gap example.

[8] and Queyranne and Sviridenko [15] (see also, for example, [3] for an application of this framework in uniformly related machines setup).

We start with a trivial upper bound 2^L on the length of an optimal schedule, where $L = \lceil \log(n \cdot \max_{j \in \mathcal{J}}(r_j + p_j)) \rceil$, and divide the time horizon into a set of geometrically increasing intervals $[1,2], (2,4], (4,8], \cdots, (2^{L-1}, 2^L]$. For the problem with only precedence and release time constraints ($P|\text{prec}, r_j| \sum_j w_j C_j$), Hall et al. argue that for each job j, if we are given the interval in which it completes in the optimal schedule, or the *completion interval* of j", we can get an approximate schedule based on this information [8]. All the jobs completing in $(2^{l-1}, 2^l]$ in the optimal schedule can be scheduled to start and complete in $(2^{l+1}, 2^{l+2}]$ using a makespan minimizing algorithm on this subset of jobs as a subroutine. This results in an 8-approximation for the weighted completion time objective. Since the completion intervals in an optimal schedule are not known, Hall et al. use an LP relaxation to obtain approximate values for the completion intervals [8]. The makespan of the makespan minimizing algorithm they use for $P|\text{prec}, r_j|C_{max}$ depends only on the Load Bound and the Chain Bound, which are both easy to bound in the same LP where they get the completion intervals. However, in our setting, we need stronger guarantees when obtaining the completion intervals with an LP than prior work provides. In particular, we need the jobs completing in a given interval to have a total resource requirement that is comparable to the resource available up to that interval. In this way, we introduce the following linear programming relaxation for the $P|$ res1, prec, $r_j| \sum_j w_j C_j$ problem which ensures that we get a good estimate on the completion interval of each job and simultaneously guarantees that the total resource requirement by jobs completing in some interval is not "too much". We let $[a]$ denote the set $\{1, 2, \cdots, a\}$ for a positive integer a.

$$\min \quad \sum_{j \in \mathcal{J}} w_j C_j \qquad \text{s.t.} \tag{LP_P}$$

$$\sum_{i=1}^{m} \sum_{t=1}^{L} x_{ijt} = 1, \qquad \forall j \in \mathcal{J}; \tag{2.1}$$

$$p_j \leq C_j - r_j, \qquad \forall j \in \mathcal{J}; \tag{2.2}$$

$$p_j \leq C_j - C_{j'}, \qquad \forall j' \prec j; \tag{2.3}$$

$$\sum_{t=1}^{L} 2^{t-1} \sum_{i=1}^{m} x_{ijt} \leq C_j, \qquad \forall j \in \mathcal{J}; \tag{2.4}$$

$$\sum_{j \in \mathcal{J}} p_j \sum_{t=1}^{l} x_{ijt} \leq 2^l, \qquad \forall i \in [m], l \in [L]; \tag{2.5}$$

$$\sum_{i=1}^{m} \sum_{t=1}^{l} x_{ijt} \leq \sum_{i=1}^{m} \sum_{t=1}^{l} x_{ij't}, \qquad \forall j' \prec j, l \in [L]; \tag{2.6}$$

$$\sum_{j \in \mathcal{J}} p_j s_j \sum_{i=1}^{m} \sum_{t=1}^{l} x_{ijt} \leq 2^l S, \qquad \forall l \in [L]; \tag{2.7}$$

$$x_{ijt} \geq 0, \qquad \forall i \in [m], j \in \mathcal{J}, t \in [L]; \tag{2.8}$$

In LP_P, we have two sets of variables: a set of real valued variables for the completion times of the jobs $\{C_j\}_{j \in \mathcal{J}}$ and a set of decision variables $\{x_{ijt}\}_{i \in [m], j \in \mathcal{J}, t \in [L]}$ that take 0-1 values in an integral solution with $x_{ijt} = 1$ implying that the job j completed on machine i in the time interval $(2^{t-1}, 2^t]$. Constraint 2.1 says a job needs to complete on some machine and in

some interval. Constraints 2.2 and 2.3 make sure that the completion times are not infeasible with respect to release times and the precedence constraints. Constraint 2.4 says that the completion time of a job needs to be later than the time point marking the start of the time interval in which the job completes. Constraint 2.5 ensures that the total processing time of the jobs completing on machine i in the first l intervals is at most the time point marking the end of the lth interval. Constraint 2.6 ensures that a job j depending on another job j' must complete in j''s completion interval or later. Finally, Constraint 2.7 is how we guarantee that the total resource requirement of the jobs completing in the first l intervals is at most the total amount of resource available in these intervals.

Let $\{\tilde{x}_{ijt}\}$ and $\{\tilde{C}_j\}$ be a solution to LP_P. We now describe how to obtain the approximate completion intervals from this fractional LP solution and the smaller problems with minimum makespan objective by partitioning the set of jobs with respect to these completion intervals. Let $\ell_1(j)$ be the first interval l where the sum of job j's variables x_{ijt} across all machines exceed $1/2$ (i.e. minimum l s.t. $\sum_{t=1}^{l}\sum_{i=1}^{m}\tilde{x}_{ijt} \geq 1/2$). Also, define $\ell_2(j)$ to be the interval containing job j's completion time (i.e. minimum l s.t. $C_j \leq 2^l$). We let $\ell(j) = \max\{\ell_1(j), \ell_2(j)\}$. Define, for each $l \in [L]$, $J_l = \{j \in \mathcal{J} : \ell(j) = l\} \subseteq \mathcal{J}$ to be the subset of jobs that "complete" in the lth interval (jobs with $\ell(j) = l$). Note that the sets $J_1, J_2, \cdots, J_L$ are disjoint and they partition the set of jobs $\mathcal{J}$.

Next, we consider each J_l as a separate instance of $P|\text{res1}, \text{prec}|C_{max}$ problem and run our makespan minimizing algorithm from Section 2.1 on the smaller instance $(J_l, m, \{p_j\}_j, S, \{s_j\}_j, \prec_{J_l})$ where $\prec_{J_l} \subseteq \prec$ is the subset of the precedence constraints that are between the jobs in J_l.

▶ **Lemma 9.** *Algorithm 1 in Section 2.1 on the instance* $(J_l, m, \{p_j\}_{j\in J_l}, S, \{s_j\}_{j\in J_l}, \prec_{J_l})$ *returns a feasible schedule of length* $(4 + 3\log(|J_l| + 1))2^l$.

Using Lemma 9, we schedule, for each $l \in [L]$, the jobs in J_l to start after $(4 + 3\log(n + 1))(1 + 2 + 4 \cdots + 2^{l-1})$ and complete before $(4 + 3\log(n + 1))(1 + 2 + 4 \cdots + 2^l)$.

▶ **Lemma 10.** *The resulting schedule satisfies resource, precedence, and release time constraints.*

The next lemma completes the proof of Theorem 8.

▶ **Lemma 11.** *The resulting schedule has weighted completion time within* $32 + 24\log(n + 1)$ *factor of the* LP_P *value* $\sum_{j\in\mathcal{J}} w_j\tilde{C}_j$.

3 Uniformly Related Machines

In this section, we describe our $O(\log n \log m)$-approximation algorithm for the problem of scheduling jobs on uniformly related machines (i.e. machines running at different speeds) under resource, precedence, and release time constraints. Again, the objective is to minimize the more general total weighted completion time. This problem is denoted as $Q|\text{res1}, \text{prec}, r_j|\sum_j w_j C_j$.

▶ **Theorem 12.** *There is an* $O(\log m \log n)$*-approximation algorithm for* $Q|res1,\ prec,\ r_j|\sum_j w_j C_j$.

Now, we have a speed f_i associated with each machine $i \in [m]$ in the input and it takes p_j/f_i amount of time to complete job $j \in \mathcal{J}$ on machine $i \in [m]$. We note that the Load, Chain, and Resource Bounds do not only depend on the set of jobs anymore, but also on

the job to machine assignments in a solution (in particular, an optimal solution). This is because the processing time p_j/f_i of a job j now depends on the machine i to which it is assigned. Note that this processing time does not change between machines running at the same speed. Thus we will be more interested in the speed that a job is assigned to than the particular machine. We let K be the number of distinct speeds and $\mathrm{f}_1, \mathrm{f}_2, \cdots, \mathrm{f}_K$ be all the distinct speeds among all m machines. We also let m_k be the number of machines with speed f_k for each $k \in [K]$. We start by solving a time-interval indexed linear program similar to the one we used in Section 2.2 but incorporates the different machine speeds in the LP.

$$\min \quad \sum_{j \in \mathcal{J}} w_j C_j \qquad \text{s.t.} \tag{LP_Q}$$

$$\sum_{k=1}^{K} \sum_{t=1}^{L} x_{kjt} = 1, \qquad \forall j \in \mathcal{J}; \tag{3.1}$$

$$\sum_{k=1}^{K} \frac{p_j}{\mathrm{f}_k} \sum_{t=1}^{L} x_{kjt} \leq C_j - r_j, \qquad \forall j \in \mathcal{J}; \tag{3.2}$$

$$\sum_{k=1}^{K} \frac{p_j}{\mathrm{f}_k} \sum_{t=1}^{L} x_{kjt} \leq C_j - C_{j'}, \qquad \forall j' \prec j; \tag{3.3}$$

$$\sum_{t=1}^{L} 2^{t-1} \sum_{k=1}^{K} x_{kjt} \leq C_j, \qquad \forall j \in \mathcal{J}; \tag{3.4}$$

$$\frac{1}{\mathrm{f}_k m_k} \sum_{j \in \mathcal{J}} p_j \sum_{t=1}^{l} x_{kjt} \leq 2^l, \qquad \forall k \in [K], l \in [L]; \tag{3.5}$$

$$\sum_{k=1}^{K} \sum_{t=1}^{l} x_{kjt} \leq \sum_{k=1}^{K} \sum_{t=1}^{l} x_{kj't}, \qquad \forall j' \prec j, l \in [L]; \tag{3.6}$$

$$\sum_{j \in \mathcal{J}} \sum_{k=1}^{K} \frac{p_j}{\mathrm{f}_k} s_j \sum_{t=1}^{l} x_{kjt} \leq 2^l S, \qquad \forall l \in [L]; \tag{3.7}$$

$$x_{kjt} \geq 0, \qquad \forall k \in [K], j \in \mathcal{J}, t \in [L]; \tag{3.8}$$

Constraints and variables of LP_Q are similar to the ones in LP_P of Section 2.2. We still have the real-valued completion time variables $\{C_j\}_{j \in \mathcal{J}}$. However, we are now interested in the distinct speed group a job is assigned to rather than the particular machine. In this way, we modify the decision variables as $\{x_{kjt}\}_{k \in [K], j \in \mathcal{J}, t \in [L]}$ where the first index k now indicates the speed group among the machines. In an integral solution, $x_{kjt} = 1$ would stand for the job j completing in the time interval $(2^{t-1}, 2^t]$ on some machine with speed f_k. Since the processing time p_j of a job j now depends on the speed it runs on, we replace p_j in the constraints by $\sum_{k=1}^{K} p_j/\mathrm{f}_k \sum_{t=1}^{L} x_{kjt}$ which essentially is the average processing time of j with respect to fractional speed assignments of j. We make the required change to Constraints 2.2, 2.3, 2.5, and 2.7 to obtain corresponding Constraints 3.2, 3.3, 3.5, and 3.7.

Similar to what we did in the identical machine setting, our algorithm will partition the set of jobs according to approximate completion intervals obtained from LP_Q. Then it will obtain a set of smaller $Q|\mathrm{res1}, \mathrm{prec}, r_j|C_{max}$ instances from the original $Q|\mathrm{res1}, \mathrm{prec}, r_j|\sum_j w_j C_j$ instance. We solve each minimum makespan instance again in two steps by considering the $Q|\mathrm{prec}, r_j|C_{max}$ instance in the first step and handling the resources in the second step. The bound on the makespan minimizing algorithm of Section 2.1 is analyzed in terms of

Load, Chain, and Resource Bounds. However, as we have indicated above, we do not have well-defined Load, Chain, and Resource Bounds in the case of machines running at different speeds. In order to use and analyze a similar two-step algorithm, we use the solution to LP_Q to first fix job to machine speed assignments, then show that the Load, Chain, and Resource Bounds under these assignments are comparable to the LP_Q value. We adapt the speed-based list scheduling algorithm of [3] as our first step for solving $Q|\text{prec}, r_j|C_{max}$ and finally we apply the Divide and Schedule procedure of Section 2.1 as our second step.

Let $\{\tilde{x}_{kjt}\}$ and $\{\tilde{C}_j\}$ be a solution to LP_Q. We obtain "completion intervals" ($\ell(j)$'s) for all the jobs and partition $\mathcal{J}$ by defining J_l for $1 \leq l \leq L$ in the same way as we did in Section 2.2. We let $\ell(j) = \max\{\ell_1(j), \ell_2(j)\}$ where $\ell_1(j)$ is defined as the minimum l s.t. $\sum_{t=1}^{l}\sum_{k=1}^{K}\tilde{x}_{kjt} \geq 1/2$ and $\ell_2(j)$ as the minimum l s.t. $\tilde{C}_j \leq 2^l$. We partition $\mathcal{J}$ by letting $J_l = \{j \in \mathcal{J} : \ell(j) = l\}$ for each $l \in [L]$.

Chudak and Shmoys [3] solve a similar LP without Constraint 3.7 to get an $O(\log m)$ approximation for $Q|\text{prec}, r_j|\sum_j w_j C_j$. They follow a similar framework by first giving a makespan minimizing algorithm for $Q|\text{prec}, r_j|C_{max}$ and then using it on J_l's as a subroutine to get the same result for the weighted completion time objective. We replace Graham's list scheduling for $P|\text{prec}|C_{max}$ with Chudak and Shmoys' speed-based list scheduling algorithm [3] for $Q|\text{prec}, r_j|C_{max}$ in our first step. We now briefly discuss their speed-based list scheduling algorithm, its analysis, and how to integrate it into our setup.

Their speed-based list scheduling algorithm uses a list to process the jobs greedily in the order given by the list similar to Graham's list scheduling. In addition, it uses a function f mapping each job j to a speed $f(j) \in \{\text{f}_1, \text{f}_2, \cdots \text{f}_K\}$. As soon as a job finishes processing, all idle machines are considered in some fixed order one by one. The algorithm considers each machine with speed f_k and schedules the first available job j in the list with $f(j) = \text{f}_k$ on this machine, where a job is available if all its predecessors are completed. They analyze this algorithm by considering the Load Bound of each speed group separately. They use an argument similar to the analysis of Graham's list scheduling by charging the busy time intervals on the Load Bounds and idle intervals on the Chain Bound. They show that the length of the makespan returned by this algorithm is at most the sum of the Load Bounds for each speed group and the Chain Bound:

$$\sum_{k=1}^{K}\frac{1}{m_k}\sum_{j:f(j)=\text{f}_k}\frac{p_j}{f(j)} + \max_{\mathcal{C} \text{ is a chain}}\sum_{j\in\mathcal{C}}\frac{p_j}{f(j)}$$

Note that the bound given above depends heavily on job to speed assignments f used by the algorithm. Given the assignments of an optimal solution, the Load Bound of each speed group and the Chain bound will be bounded by the optimal makespan length. This would put the makespan of the algorithm within a factor $K+1$ of the optimal solution length. Since we do not have the optimal assignments, we now describe how to get approximate job to speed assignments f by applying standard filtering techniques on the fractional solution $\{\tilde{x}_{kjt}\}$ to LP_Q. Consider a partition J_l and a job $j \in J_l$. Let $\alpha_j = \sum_{i=1}^{K}\sum_{t=1}^{l}\tilde{x}_{kjt}$ and let $\overline{x}_{kj} = \sum_{t=1}^{l}\tilde{x}_{kjt}/\alpha_j$. Note $\alpha_j \geq 1/2$ by definition of $\ell(j) = l$. Let the average processing time of a job j in LP_Q solution be $p_j^{avg} = \sum_{k=1}^{K}(p_j/\text{f}_k)\overline{x}_{kj}$. We define $f(j)$ to be the speed of the maximum capacity speed group among all speeds more than half the average speed that j is assigned to in the fractional LP_Q solution.[3] Formally, let

[3] Equivalently, since p_j is a constant in the evaluation of $f(j)$, we can define it to be the f_k that maximizes $\text{f}_k m_k$.

$$f(j) = \underset{\mathrm{f}_k:\ p_j/\mathrm{f}_k \le 2p_j^{avg}}{\arg\min} \frac{p_j}{\mathrm{f}_k m_k}.$$

Similar to [3], we show that the sum of the Load Bounds of all speed groups under these job to speed assignments given by f is bounded by $4K \cdot 2^l$:

▶ **Lemma 13.**

$$\sum_{k=1}^{K} \frac{1}{m_k} \sum_{j \in J_l:\ f(j)=\mathrm{f}_k} \frac{p_j}{\mathrm{f}_k} \le 4K \cdot 2^l.$$

Now we show that the total processing time of any chain in J_l under the assignments given by f is also bounded:

▶ **Lemma 14.** *For any chain $\mathcal{C}$ of precedence constraints from $\prec_{J_l}$, we have $\sum_{j \in \mathcal{C}} p_j/f(j) \le 4 \cdot 2^l$.*

Lemmas 13 and 14 show that the speed-based list scheduling algorithm we employ in the first step returns a schedule of length $O(K) \cdot 2^l$ on the smaller instance we get from the partition J_l.

After getting the intermediate schedule from the first step of our algorithm, we run our second step (Divide and Schedule) respecting job to machine assignments of the intermediate schedule. Note that J_{mid} in **DS** has only one job per machine because jobs in J_{mid} all run in parallel at some point in the intermediate schedule. Thus, we do not need any modification to the **NFDH** subroutine to ensure the same job to machine assignments in the final schedule.

Next, we show that Resource Bound of the sub instance obtained from the partition J_l is bounded by $4S \cdot 2^l$ under the same job to speed assignments f.

▶ **Lemma 15.** $\mathsf{RB}(J_l) \le 4S \cdot 2^l$.

Our second step returns a schedule of length bounded by $\mathbf{DS}(J, B, E) \le 2\mathsf{RB}(J)/S + (E - B)\log(|J| + 1)$ where $(E - B)$ is the length of the intermediate schedule obtained by the first step. Lemmas 13, 14, and 15 prove that, for any $l \in [L]$, this bound is at most $O(K \log n) \cdot 2^l$ on the sub-instance obtained from the jobs in J_l.

As we did in Section 2.2, we schedule the jobs in J_l for each $l \in [L]$ to start after $O(K \log n)(1 + 2 + 4 \cdots + 2^l)$ and complete before $O(K \log n)(1 + 2 + 4 \cdots + 2^l + 2^{l+1})$. This again gives us a feasible schedule as in Lemma 10 and, given that $2^{l-2} \le \tilde{C}_j$ as in the proof of Lemma 11, we have an $O(K \log n)$ approximation.

Finally using the preprocessing of Chudak and Shmoys [3], we can assume that we only have $K = \log m$ distinct speed groups. This comes with a loss of an additional constant factor on the approximation ratio of our algorithm for the minimum makespan objective. We first discard all the machines with speed less than $1/m$ times the speed of the fastest machine and then round down each remaining speed to the nearest power of $1/2$. Discarding slow machines only increases the optimal makespan by a factor of 2 because we discard no more than m machines each of which is at most as fast as $1/m$ times the speed of the fastest machine. This means the fastest machine can process the jobs of the discarded machines with a loss of factor 2 in the length of the schedule. Similarly, we only lose another factor of 2 by rounding down the speeds of the remaining machines. Since we have at most $\log m$ distinct speeds after the preprocessing, the algorithm described above on each $Q|\mathrm{res1}, \mathrm{prec}, r_j|C_{max}$ sub-instance is an $O(\log m \log n)$-approximation with $K = \log m$.

References

1. John Augustine, Sudarshan Banerjee, and Sandy Irani. Strip packing with precedence constraints and strip packing with release times. In *Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '06, pages 180–189, New York, NY, USA, 2006. ACM.
2. Abbas Bazzi and Ashkan Norouzi-Fard. Towards tight lower bounds for scheduling problems. In *Proceedings of the 23rd Annual European Symposium on Algorithms, ESA'15*, volume 9294, page 118. Springer, 2015.
3. Fabián A. Chudak and David B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '97, pages 581–590, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.
4. Jr. E. G. Coffman, M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980.
5. M. R. Garey and R. L. Grahams. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4:187–200, 1975.
6. R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979. Discrete Optimization II.
7. Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell Labs Technical Journal*, 45(9):1563–1581, 1966.
8. Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22(3):513–544, 1997.
9. Klaus Jansen, Marten Maack, and Malin Rau. Approximation schemes for machine scheduling with resource (in-)dependent processing times. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 1526–1542, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics.
10. Peter Kogge, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, Katherine Yelick, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Stephen Keckler, Dean Klein, Peter Kogge, R. Stanley Williams, and Katherine Yelick. Exascale computing study: Technology challenges in achieving exascale systems, 2008.
11. Shi Li. Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations. In *IEEE 57th Annual Symposium on Foundations of Computer Science*, FOCS '17, 2017.
12. Alix Munier, Maurice Queyranne, and Andreas Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. In *Integer Programming and Combinatorial Optimization*, IPCO '98, 1998.
13. Martin Niemeier and Andreas Wiese. Scheduling with an orthogonal resource constraint. In *10th Workshop on Approximation and Online Algorithms (WAOA2012)*, number EPFL-CONF-181146, 2012.
14. Maurice Queyranne and Andreas S. Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. *SIAM J. Comput.*, 35(5):1241–1253, 2006.

15 Maurice Queyranne and Maxim Sviridenko. Approximation algorithms for shop scheduling problems with minsum objective. *Journal of Scheduling*, 5(4):287–305, 2002.

16 Vivek Sarkar, Saman Amarasinghe, Dan Campbell, William Carlson, Andrew Chien, William Dally, Elmootazbellah Elnohazy, Mary Hall, Robert Harrison, William Harrod, Kerry Hill, Jon Hiller, Sherman Karp, Charles Koelbel, David Koester, Peter Kogge, John Levesque, Daniel Reed, Robert Schreiber, Mark Richards, Al Scarpelli, John Shalf, Allan Snavely, and Thomas Sterling. Exascale software study: Software challenges in extreme scale systems, 2009. DARPA IPTO Study Report for William Harrod.

17 Ola Svensson. Conditional hardness of precedence constrained scheduling on identical machines. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 745–754, New York, NY, USA, 2010. ACM.

18 ExaOSR Team. Exascale operating systems and runtime software report. Online document, `https://science.energy.gov/~/media/ascr/pdf/research/cs/Exascale%20Workshop/ExaOSR-Report-Final.pdf`.

Parameterized Approximation Schemes for Steiner Trees with Small Number of Steiner Vertices

Pavel Dvořák
Computer Science Institute, Charles University, Prague, Czechia
koblich@iuuk.mff.cuni.cz

Andreas Emil Feldmann
Department of Applied Mathematics, Charles University, Prague, Czechia
feldmann.a.e@gmail.com

Dušan Knop
Department of Informatics, University of Bergen, Bergen, Norway and
Department of Applied Mathematics, Charles University, Prague, Czechia
knop@kam.mff.cuni.cz

Tomáš Masařík
Department of Applied Mathematics, Charles University, Prague, Czechia
masarik@kam.mff.cuni.cz

Tomáš Toufar
Computer Science Institute, Charles University, Prague, Czechia
toufar@iuuk.mff.cuni.cz

Pavel Veselý
Computer Science Institute, Charles University, Prague, Czechia
vesely@iuuk.mff.cuni.cz

Abstract

We study the Steiner Tree problem, in which a set of *terminal* vertices needs to be connected in the cheapest possible way in an edge-weighted graph. This problem has been extensively studied from the viewpoint of approximation and also parametrization. In particular, on one hand Steiner Tree is known to be APX-hard, and W[2]-hard on the other, if parameterized by the number of non-terminals (*Steiner vertices*) in the optimum solution. In contrast to this we give an *efficient parameterized approximation scheme* (EPAS), which circumvents both hardness results. Moreover, our methods imply the existence of a *polynomial size approximate kernelization scheme* (PSAKS) for the considered parameter.

We further study the parameterized approximability of other variants of Steiner Tree, such as Directed Steiner Tree and Steiner Forest. For neither of these an EPAS is likely to exist for the studied parameter: for Steiner Forest an easy observation shows that the problem is APX-hard, even if the input graph contains no Steiner vertices. For Directed Steiner Tree we prove that computing a constant approximation for this parameter is W[1]-hard. Nevertheless, we show that an EPAS exists for Unweighted Directed Steiner Tree. Also we prove that there is an EPAS and a PSAKS for Steiner Forest if in addition to the number of Steiner vertices, the number of connected components of an optimal solution is considered to be a parameter.

2012 ACM Subject Classification Mathematics of computing → Combinatorics, Mathematics of computing → Graph theory, Theory of computation → Design and analysis of algorithms

Keywords and phrases Steiner Tree, Steiner Forest, Approximation Algorithms, Parameterized Algorithms, Lossy Kernelization

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.26

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 26; pp. 26:1–26:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version The full preprinted version of this article is available at `https://arxiv.org/abs/1710.00668v2`, [17].

Funding This work was partially supported by the project SVV–2017–260452. D. Knop, T. Masařík and P. Veselý were supported by project 17-09142S of GAČR. D. Knop was supported by the NFR MULTIVAL project. A. E. Feldmann was supported by the Czech Science Foundation GAČR (grant #17-10090Y). A.E. Feldmann, T. Masařík, T. Toufar, and P. Veselý were supported by the project GAUK 1514217. P. Dvořák has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 616787.

Acknowledgements We would like to thank Michael Lampis and Édouard Bonnet for helpful discussions on the problem. Also we thank Jiří Sgall and Martin Böhm for finding a mistake in the proof of Lemma 9.

1 Introduction

In this paper we study several variants of the Steiner Tree problem. In its most basic form this optimization problem takes an undirected graph $G = (V, E)$ with edge weights $w(e) \in \mathbb{R}_0^+$ for every $e \in E$, and a set $R \subseteq V$ of *terminals* as input. The non-terminals in $V \setminus R$ are called *Steiner vertices*. A *Steiner tree* is a tree in the graph G, which spans all terminals in R and may contain some of the Steiner vertices. The objective is to minimize the total weight $\sum_{e \in E(T)} w(e)$ of the computed Steiner tree $T \subseteq G$. This fundamental optimization problem is one of the 21 original NP-hard problems listed by Karp [25] in his seminal paper from 1972, and has been intensively studied since then. The Steiner Tree problem and its variants have applications in network design, circuit layouts, and phylogenetic tree reconstruction, among others (see survey [23]).

Two popular ways to handle the seeming intractability of NP-hard problems are to design *approximation* [35] and *parameterized* [12] algorithms. For the former, an *α-approximation* is computed in polynomial time for some factor α specific to the algorithm, i.e., the solution is always at most a multiplicative factor of α worse than the optimum of the input instance. The Steiner Tree problem, even in its basic form as defined above, is APX-hard [11], i.e., it is NP-hard to obtain an approximation factor of $\alpha = \frac{96}{95} \approx 1.01$. However a factor of $\alpha = \ln(4) + \varepsilon \approx 1.39$ can be achieved in polynomial time [5], which is the currently best factor known for this runtime.

For parameterized algorithms, an instance is given together with a *parameter* p describing some property of the input. The optimum solution is computed in time $f(p) \cdot n^{\mathrm{O}(1)}$, where f is a computable function independent of the input size n. If such an algorithm exists, we call the problem *fixed-parameter tractable* (FPT) for parameter p. A well-studied parameter for the Steiner Tree problem is the number of terminals $|R|$. It is known since the classical result of Dreyfus and Wagner [16] that the Steiner Tree problem is FPT for this parameter, as the problem can be solved in time $3^{|R|} \cdot n^{\mathrm{O}(1)}$ if $n = |V|$. This can be improved to $2^{|R|} \cdot n^{\mathrm{O}(1)}$ if the input graph is unweighted using the results of Björklund et al. [2]. A somewhat complementary and less-studied parameter to the number of terminals is the number of Steiner vertices in the optimum solution, i.e., $p = |V(T) \setminus R|$ if T is an optimum Steiner tree. It is known [15] that Steiner Tree is W[2]-hard for parameter p and therefore is unlikely to be FPT, in contrast to the parameter $|R|$. This parameter p has been mainly studied in the context of unweighted graphs before. The problem remains W[2]-hard in this special case and therefore the focus has been on designing parameterized algorithms for restricted graph classes, such as planar or d-degenerate graphs [24, 33].

In contrast to this, our question is: what can be done in the most general case, in which the class of input graphs is unrestricted and edges may have weights? Our main result is that we can overcome the APX-hardness of Steiner Tree on one hand, and on the other hand also the W[2]-hardness for our parameter of choice p, by combining the two paradigms of approximation and parametrization. This relatively new and growing area has gained quite a bit of attention recently (see e.g., [3, 6, 8, 9, 10, 18, 20, 26, 27, 28, 29, 32, 34]). We show that there is an *efficient parameterized approximation scheme* (EPAS), which for any $\varepsilon > 0$ computes a $(1+\varepsilon)$-approximation in time $f(p,\varepsilon) \cdot n^{\mathrm{O}(1)}$ for a computable function f independent of n. Note that here we consider the approximation factor of the algorithm as a parameter as well, which accounts for the "efficiency" of the approximation scheme (analogous to an *efficient polynomial time approximation scheme* or EPTAS). In fact, as summarized in the following theorem, our algorithm computes an approximation to the cheapest tree having at most p Steiner vertices, even if better solutions with more Steiner vertices exist.

▶ **Theorem 1.** *There is an algorithm for* Steiner Tree*, which given an edge-weighted undirected graph $G = (V, E)$, terminal set $R \subseteq V$, $\varepsilon > 0$, and integer p, computes a $(1+\varepsilon)$-approximation to the cheapest Steiner tree $T \subseteq G$ with $p \geq |V(T) \setminus R|$ in time $2^{\mathrm{O}(p^2/\varepsilon^4)} \cdot n^{\mathrm{O}(1)}$.* [1]

Many variants of the Steiner Tree problem exist, and we explore the applicability of our techniques to some common ones. For the Directed Steiner Tree problem the aim is to compute an *arborescence*, i.e., a directed graph obtained by orienting the edges of a tree so that exactly one vertex called the *root* has in-degree zero (which means that all vertices are reachable from the root). More concretely, the input consists of a directed graph $G = (V, A)$ with arc weights $w(a) \in \mathbb{R}_0^+$ for every $a \in A$, a terminal set $R \subseteq V$, and a specified terminal $r \in R$. A *Steiner arborescence* is an arborescence in G with root r containing all terminals R. The objective is to find a Steiner arborescence $T \subseteq G$ minimizing the weight $\sum_{a \in A(T)} w(a)$. This problem is notoriously hard to approximate: no $\mathrm{O}\big(\log^{2-\varepsilon}(n)\big)$-approximation exists unless NP $\subseteq$ ZTIME$(n^{\mathrm{polylog}(n)})$ [22]. But even for the Unweighted Directed Steiner Tree problem in which each arc has unit weight, a fairly simple reduction from the Set Cover problem implies that no $((1-\varepsilon)\ln n)$-approximation algorithm is possible unless P = NP [13, 22]. At the same time, even Unweighted Directed Steiner Tree is W[2]-hard for our considered parameter p [24, 30], just as for the undirected case. For this reason, all previous results have focused on restricted inputs: Jones et al. [24] prove that when combining the parameter p with the size of the largest excluded topological minor of the input graph, Unweighted Directed Steiner Tree is FPT. They also show that if the input graph is acyclic and d-degenerate, the problem is FPT for the combined parameter p and d.

Our focus again is on general unrestricted inputs. We are able to leverage our techniques to the unweighted directed setting, and obtain an EPAS, as summarized in the following theorem. Here the cost of a Steiner arborescence is the number of contained arcs.

▶ **Theorem 2.** *There is an algorithm for* Unweighted Directed Steiner Tree*, which given an unweighted directed graph $G = (V, A)$, terminal set $R \subseteq V$, root $r \in R$, $\varepsilon > 0$, and integer p, computes a $(1+\varepsilon)$-approximation to the cheapest Steiner arborescence $T \subseteq G$ with $p \geq |V(T) \setminus R|$ in time $2^{p^2/\varepsilon} \cdot n^{\mathrm{O}(1)}$.* [1]

[1] If the input to this optimization problem is malformed (e.g., if p is smaller than the number of Steiner vertices of any feasible solution) then the output of the algorithm can be arbitrary (cf. [28])

Can our techniques be utilized for the even more general case when arcs have weights? Interestingly, in contrast to the above theorem we can show that in general the Directed Steiner Tree problem most likely does not admit such approximation schemes, even when allowing "non-efficient" runtimes of the form $f(p,\varepsilon) \cdot n^{g(\varepsilon)}$ for any computable functions f and g. This follows from the next theorem, since setting ε to any constant, the existence of such a $(1+\varepsilon)$-approximation algorithm would imply $\mathsf{W}[1] = \mathsf{FPT}$.

▶ **Theorem 3.** *For any constant α, it is $\mathsf{W}[1]$-hard to compute an α-approximation of the optimum Steiner arborescence T for* Directed Steiner Tree *parameterized by $|V(T) \setminus R|$, if the input graph is arc-weighted.*

Other common variants of Steiner Tree include the Prize Collecting Steiner Tree and Steiner Forest problems. The latter takes as input an edge-weighted undirected graph $G = (V, E)$ and a list $\{s_1, t_1\}, \ldots, \{s_k, t_k\}$ of terminal pairs, i.e., $R = \{s_i, t_i \mid 1 \leq i \leq k\}$. A *Steiner forest* is a forest F in G for which each $\{s_i, t_i\}$ pair is in the same connected component, and the objective is to minimize the total weight of the forest F. For this variant it is not hard to see that parametrizing by $p = |V(F) \setminus R|$ cannot yield any approximation scheme, as a simple reduction from Steiner Tree shows that the problem is APX-hard even if the input has no Steiner vertices (cf. [17]). For the Prize Collecting Steiner Tree problem, the input is again a terminal set in an edge-weighted graph, but the terminals have additional costs. A solution tree is allowed to leave out a terminal but has to pay its cost in return (cf. [35]). It is also not hard to see that this problem is APX-hard, even if there are no Steiner vertices at all. These simple results show that our techniques to obtain approximation schemes reach their limit quite soon: with the exception of Unweighted Directed Steiner Tree, most common variants of Steiner Tree seem not to admit approximation schemes for our parameter p. We are however able to generalize our EPAS to Steiner Forest if we combine p with the number c of connected components in the optimum solution. In fact, our main result of Theorem 1 is a corollary of the next theorem, using only the first part of the above mentioned reduction from Steiner Tree. Due to this, it is not possible to have a parameterized approximation scheme for the parameter c alone, as such an algorithm would imply a polynomial time approximation scheme for the APX-hard Steiner Tree problem. Hence the following result necessarily needs to combine the parameters p and c.

▶ **Theorem 4.** *There is an algorithm for* Steiner Forest*, which given an edge-weighted undirected graph $G = (V, E)$, a list $\{s_1, t_1\}, \ldots, \{s_k, t_k\} \subseteq V$ of terminal pairs, $\varepsilon > 0$, and integers p, c, computes a $(1+\varepsilon)$-approximation to the cheapest Steiner forest $F \subseteq G$ with at most c connected components and $p \geq |V(F) \setminus R|$ where $R = \{s_i, t_i \mid 1 \leq i \leq k\}$, in time $(2c)^{\mathrm{O}\left((p+c)^2/\varepsilon^4\right)} \cdot n^{\mathrm{O}(1)}$.* [1]

A topic tightly connected to parameterized algorithms is kernelization. We here use the framework of Lokshtanov et al. [28], who also give a thorough introduction to the topic. Loosely speaking, a *kernelization algorithm* runs in polynomial time, and, given an instance of a parameterized problem, computes another instance of the same problem, such that the size of the latter instance is at most $f(p)$ for some computable function f in the parameter p of the input instance. The computed instance is called the *kernel*, and for an optimization problem it must be possible to efficiently convert an optimum solution to the kernel into an optimum solution to the input instance.

A fundamental result of parameterized complexity says that a problem is FPT if and only if it has a kernelization algorithm [12]. This means that for our parameter p, most likely

Steiner Tree does not have a kernelization algorithm, as it is W[2]-hard. For this reason, the focus of kernelization results have previously again shifted to special cases. By a folklore result, Steiner Tree is FPT for our parameter p if the input graph is planar (cf. [24]). Of particular interest are *polynomial kernels*, which have size polynomial in the input parameter. The idea is that computing the kernel in this case is an efficient preprocessing procedure for the problem, such that exhaustive search algorithms can be used on the kernel. Suchý [33] proved that Unweighted Steiner Tree parameterized by p admits a polynomial kernel if the input graph is planar.

Our aspirations again are to obtain results for inputs that are as general as possible, i.e., on unrestricted edge-weighted input graphs. We prove that Steiner Tree has a polynomial *lossy* (approximate) kernel, despite the fact that the problem is W[2]-hard: an *α-approximate kernelization algorithm* is a kernelization algorithm that computes a new instance for which a given β-approximation can be converted into an $\alpha\beta$-approximation for the input instance in polynomial time. The new instance is now called a *(polynomial) approximate kernel*, and its size is again bounded as a function (a polynomial) of the parameter of the input instance.

Just as for our parameterized approximation schemes in Theorems 1 and 4, we prove the existence of a lossy kernel for Steiner Tree by a generalization to Steiner Forest where we combine the parameter p with the number c of connected components in the optimum solution. Also, our lossy kernel can approximate the optimum arbitrarily well: we prove that for our parameter the Steiner Forest problem admits a *polynomial size approximate kernelization scheme* (PSAKS), i.e., for every $\varepsilon > 0$ there is a $(1+\varepsilon)$-approximate kernelization algorithm that computes a polynomial approximate kernel. An easy corollary then is that Steiner Tree parameterized only by p also has a PSAKS, by setting $c = 1$ in Theorem 5 and using the reduction from Steiner Tree to Steiner Forest as above.

▶ **Theorem 5.** *There is a $(1+\varepsilon)$-approximate kernelization algorithm for* Steiner Forest*, which given an edge-weighted undirected graph $G = (V, E)$, a list $\{s_1, t_1\}, \ldots, \{s_k, t_k\} \subseteq V$ of terminal pairs, and integers p, c, computes an approximate kernel of size $((p+c)/\varepsilon)^{2^{O(1/\varepsilon)}}$, if for the optimum Steiner forest $F \subseteq G$, $p \geq |V(F) \setminus R|$ where $R = \{s_i, t_i \mid 1 \leq i \leq k\}$, the number of connected components of F is at most c, and $\varepsilon > 0$.* [1]

Analogous to approximation schemes, it is possible to distinguish between efficient and non-efficient kernelization schemes: a PSAKS is *size efficient* if the size of the approximate kernel is bounded by $f(\varepsilon) \cdot p^{O(1)}$, where p is the parameter and f is a computable function independent of p. Our bound on the approximate kernel size in Theorem 5 implies that we do *not* obtain a size efficient PSAKS for either Steiner Forest or Steiner Tree. This is in contrast to the existence of efficient approximation schemes for the same parameters in Theorems 1 and 4. We leave open whether a size efficient PSAKS can be found in either case. Interestingly, we also do not obtain any PSAKS for the Unweighted Directed Steiner Tree problem, even though by Theorem 2 an EPAS exists. We leave open whether a PSAKS can be found for this variant as well.

Used techniques. Our algorithms are based on the intuition that a Steiner tree containing only few Steiner vertices but many terminals must either contain a large component induced by terminals, or a Steiner vertex with many terminal neighbors forming a large star. A high-level description of our algorithms for Unweighted Directed Steiner Tree and Steiner Forest therefore is as follows. In each step a tree is found in the graph in polynomial time, which connects some terminals using few Steiner vertices. We save this tree as part of the approximate solution and then contract it in the graph. The vertex resulting from

the contraction is declared a terminal and the process repeats for the new graph. Previous results [24, 33] have also built on this straightforward procedure in order to obtain FPT algorithms and polynomial kernels for special cases of Unweighted Directed Steiner Tree and Unweighted Steiner Tree. In particular, in the unweighted undirected setting it is a well-known fact (cf. [33]) that contracting an adjacent pair of terminals is always a safe option, as there always exists an optimum Steiner tree containing this edge. However this immediately breaks down if the input graph is edge-weighted, as an edge between terminals might be very costly and should therefore not be contained in any (approximate) solution.

Instead we employ more subtle contraction rules, which use the following intuition. Every time we contract a tree with ℓ terminals we decrease the number of terminals by $\ell - 1$ (as the vertex arising from a contraction is a terminal). Our ultimate goal would be to reduce the number of terminals to one—at this point, the edges that we contracted during the whole run connect all the terminals. Decreasing the number of terminals by one can therefore be seen as a "unit of work". We will pick a tree with the lowest cost per unit of work done, and prove that as long as there are sufficiently many terminals left in the graph, these contractions only lose an ε-factor compared to the optimum. As soon as the number of terminals falls below a certain threshold depending on the given parameter, we can use an FPT algorithm computing the optimum solution in the remaining graph. This algorithm is parametrized by the number of terminals, which now is bounded by our parameter. For the variants of Steiner Tree considered in our positive results, such FPT algorithms can easily be obtained from the ones for Steiner Tree [16, 2]. Adding this exact solution to the previously contracted trees gives a feasible solution that is a $(1 + \varepsilon)$-approximation.

Each step in which a tree is contracted in the graph, can be seen as a *reduction rule* as used for kernelization algorithms. Typically, a proof for a kernelization algorithm will define a set of reduction rules and then show that the instance resulting from applying the rules exhaustively has size bounded as a function in the parameter. To obtain an α-approximate kernelization algorithm, additionally it is shown that each reduction rule is *α-safe*. Roughly speaking, this means that at most a factor of α is lost when applying any number of α-safe reduction rules.

Contracting edges in a directed graph may introduce new paths, which did not exist before. Therefore, for the Unweighted Directed Steiner Tree problem, we need to carefully choose the arborescence to contract. In order to prove Theorem 2 we show that each contraction is a $(1+\varepsilon)$-safe reduction rule. However, the total size of the graph resulting from exhaustively applying the contractions is not necessarily bounded as a function of our parameter. Thus we do not obtain an approximate kernel.

For Steiner Forest the situation is in a sense the opposite. Choosing a tree to contract follows a fairly simple rule. On the downside however, the contractions we perform are not necessarily $(1 + \varepsilon)$-safe reduction rules. In fact there are examples in which a single contraction will lose a large factor compared to the optimum cost. We are still able to show however, that after performing all contractions exhaustively, any β-approximation to the resulting instance can be converted into a $(1 + \varepsilon)\beta$-approximation to the original input instance. Even though the total size of the resulting instance again cannot be bounded in terms of our parameter, for Steiner Forest we can go on to obtain a PSAKS. For this we utilize a result of Lokshtanov et al. [28], which shows how to obtain a PSAKS for Steiner Tree if the parameter is the number of terminals. This result can be extended to Steiner Forest, and since our instance has a number of terminals bounded in our parameter after applying all contractions, we obtain Theorem 5.

Finally, to obtain our inapproximability result of Theorem 3, we use a reduction from the Dominating Set problem. It was recently shown by Chen and Lin [8] that this problem does not admit parameterized α-approximation algorithms for any constant α, if the parameter is the solution size, unless W[1] = FPT. We are able to exploit this to also show that no such algorithm exists for Directed Steiner Tree with edge weights, under the same assumption.

All missing proofs of this paper are deferred to the full version of the paper [17].

Related work. As the Steiner Tree problem and its variants have been studied since decades, the literature on this topic is huge. We only present a selection of related work here, that was not yet mentioned above.

For planar graphs [4] it was shown that an EPTAS exists for Steiner Tree. For Steiner Forest a 2-approximation can be computed in polynomial time on general inputs [1], but an EPTAS also exists if the input is planar [19]. If the Unweighted Steiner Tree problem is parametrized by the solution size, it is known [14] that no polynomial (exact) kernel exists, unless NP $\subseteq$ coNP/poly. If the input is restricted to planar or bounded-genus graphs it was shown that polynomial kernels do exist for this parametrization [31]. It was later shown [33] that for planar graphs this is even true for our parameter p. For the Directed Steiner Tree problem it is a long standing open problem whether a polylogarithmic approximation can be computed in polynomial time. It is known that an $O(|R|^{\varepsilon})$-approximation can be computed in polynomial time [7], and an $O(\log^2 n)$-approximation in quasi-polynomial time [7]. A recent result [21] considers generalizations of Directed Steiner Tree and characterizes which of these problems are FPT and which are W[1]-hard for parameter $|R|$.

2 The weighted undirected Steiner forest and Steiner tree problems

In this section we describe an approximate polynomial time preprocessing algorithm that returns an instance of Steiner Forest containing at most $O((p+c)^2/\varepsilon^4)$ terminals if there is a Steiner forest with at most p Steiner vertices and at most c connected components. We can use this algorithm in two ways. Either we can proceed with a kernelization derived from Lokshtanov et al. [28] and obtain a polynomial size lossy kernel (Theorem 5), or we can run an exact FPT algorithm derived from Dreyfus and Wagner [16] on the reduced instance, obtaining an EPAS running in single exponential time with respect to the parameters (Theorems 1 and 4). In both cases we use the combined parameter (p, c).

We first rescale all weights so that every edge has weight strictly greater than 1. Then, in each step of our algorithm we pick a star, add it to the solution, and contract the star in the current graph. We repeat this procedure until the number of terminals falls below a specified bound depending on ε, p, and c. To describe how we pick the star to be contracted in each step, we need to introduce the *ratio* of a star. Let C be a set of edges of a star, i.e., all edges of C are incident to a common vertex which is the *center* of the star, and denote by Q the set of terminals incident to C. Provided $|Q| \geq 2$, we define the *ratio* of C as $w(C)/(|Q|-1)$, where $w(C) = \sum_{e \in C} w(e)$. Note that we allow C to contain only a single edge if it joins two terminals. Observe also that due to rescaling of edge weights each star has ratio strictly greater than 1.

In every step, our algorithm contracts a star with the best available ratio (i.e., the lowest ratio among all stars connecting at least two terminals). Due to the following lemma, a star with the best ratio has a simple form: it consists of the cheapest i edges incident to its center vertex and some terminal. As there are n possible center vertices and at most n incident edges to each center, the best ratio star can be found in time $O(n^2)$.

▶ **Lemma 6.** *Let v be a vertex and denote by $q_1, q_2, \ldots$ the terminals adjacent to v, where $w(vq_1) \leq w(vq_2) \leq \cdots$, i.e., the terminals are ordered non-decreasingly by the weight of the corresponding edge vq_i. The star with the best ratio having v as its center has edge set $\{vq_1, vq_2, \ldots, vq_\ell\}$ for some ℓ.*

To analyse our algorithm we need to keep track of the different graphs resulting from each contraction step t. Initially we set G_0 to the input graph, and in each step $t \geq 0$ we obtain a new graph G_{t+1} from G_t by contracting a set of edges C_t in G_t, such that C_t forms a star of minimum ratio in G_t. That is, we obtain G_{t+1} from G_t by identifying all vertices incident to edges in C_t, removing all resulting loops, and among the resulting parallel edges we delete all but the lightest one with respect to their weights. We also adjust the terminal pairs in a natural way: let v be the vertex of G_{t+1} resulting from contracting C_t. If G_t had a terminal pair $\{s,t\}$ such that s is incident to some edge of C_t while t is not, then we introduce the terminal pair $\{v,t\}$ for G_{t+1}. Also every terminal pair $\{s,t\}$ of G_t for which neither s nor t is incident to any edge of C_t is introduced as a terminal pair of G_{t+1}. Any terminal pair for which both s and t are incident to edges of C_t is going to be connected by a path in the computed solution, as it will contain C_t. Hence, such a terminal pair can be safely removed.

The algorithm stops contracting best-ratio stars when there are less than τ terminals left; the exact value of τ depends on p, c, and the desired approximation factor and we specify it later. If this happens in step $\tilde{t}$, the solution lifting algorithm takes a feasible solution F of $G_{\tilde{t}}$ and returns the union of F and $\bigcup_{t=0}^{\tilde{t}} C_t$. Such a solution is clearly feasible, since we adapted the terminal pairs accordingly after each contraction.

For the purpose of analysis, we consider a solution in the current graph G_t that originates from a solution of the original instance G_0, but may contain edges that are heavier than those in G_t. More concretely, denote by F_0^* a solution in G_0 with at most p Steiner vertices and at most c components, i.e., F_0^* is a Steiner forest containing every s_i-t_i path. We remark that F_0^* may or may not be an optimum solution of G_0. Given F_t^* for $t \geq 0$, we modify this solution to obtain a new feasible solution F_{t+1}^* on the terminal pairs of G_{t+1}. Note that the edges of the contracted star C_t might not be part of F_t^*. We still mimic the contraction of the star in F_t^*: to obtain F_{t+1}^* from F_t^*, we identify all leaves of C_t (which are terminals by Lemma 6 and thus part of the solution F_t^*) and possibly also the center v of C_t if it is in F_t^*. This results in a vertex v'. We now want to delete edges incident to v' in such a way that we are left with an acyclic feasible solution. If we delete an inclusion-wise minimal feedback edge set, we clearly get a feasible solution. Let Q_t denote the set of terminals incident to C_t. We choose a feedback edge set D_t for which every edge was incident to a vertex of Q_t before the contraction in F_t^*, i.e., an edge of G_t corresponding to an edge of D_t never connects two Steiner vertices. Note that such an inclusion-wise minimal feedback edge set always exists: if we delete all edges of F_t^* incident to Q_t except C_t and then contract C_t, we get an acyclic graph. See Figure 1 for an illustration.

The resulting graph is F_{t+1}^*, which now forms a forest connecting all terminal pairs of G_{t+1}. Note that for each edge in F_{t+1}^* there is a corresponding edge in G_{t+1}, which however may be lighter in G_{t+1}, as from each bundle of parallel edges in G_t we keep the lightest one, but this edge may not exist in F_t^*.

To show that our algorithm only loses an ε-factor compared to the cost of the solution F_0^*, we will compare the cost of the edges C_t contracted by our algorithm to the set $D_t = E(F_{t+1}^*) \setminus E(F_t^*)$ of deleted edges of F_t^*. Note that there are at least $|Q_t| - c$ edges in D_t, since we contracted Q_t terminals in the forest F_t^* with at most c connected components to obtain F_{t+1}^*, and a forest on n vertices and k components has $n - k$ edges. We decrease the number of vertices of F_t^* by at least $|Q_t| - 1$ (one more if the center of the star with edge

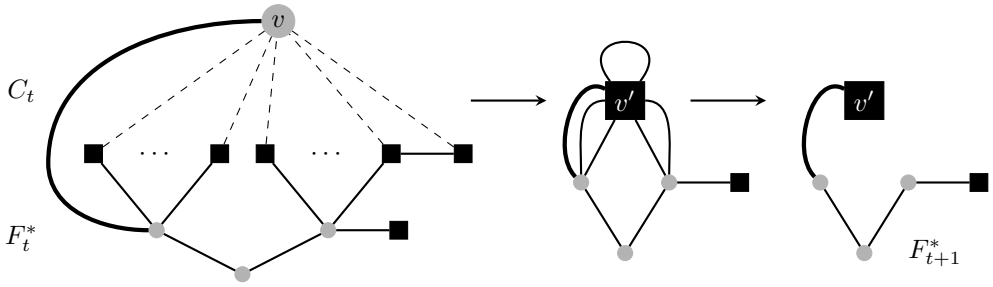


Figure 1 An example of creating F_{t+1}^* from F_t^* after a contraction C_t. Each edge in C_t (dashed) may or may not be in F_t^*. The thick edge cannot be in D_t because it is not incident to any terminal.

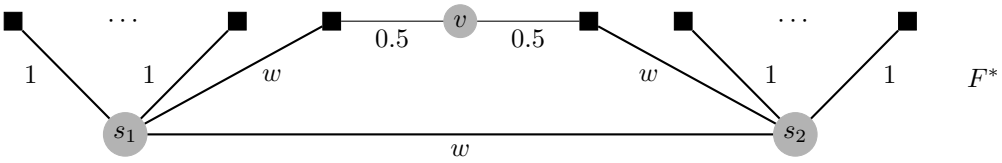


Figure 2 An example of a bad contraction. The numbers of terminals can be arbitrarily large and the weight w can be arbitrarily small. The star centered at v has ratio 1 while every star centered either at s_1 or s_2 has ratio slightly more than 1. By contracting the star centered at v we create a cycle containing only edges of weight w. Thus, for a sufficiently small value of w the contraction cannot be charged.

set C_t was a Steiner vertex present in F_t^*), and we decrease the number of components by at most $c-1$. Note also that for any two time steps $t \neq t'$, the sets D_t and $D_{t'}$, but also the sets C_t and $C_{t'}$, are disjoint. Thus if $w(C_t) \leq (1+\varepsilon)w(D_t)$ for every t, then our algorithm computes a $(1+\varepsilon)$-approximation. Unfortunately, this is not always the case: there are contractions for which this condition does not hold (see Figure 2) and we have to account for them differently.

▶ **Definition 7.** If $w(C_t) \leq (1+\varepsilon)w(D_t)$ we say that the contracted edge set C_t in step t is *good*; otherwise C_t is *bad*. Moreover, if F_t^* has strictly more components than F_{t+1}^*, we say that C_t is *multiple-component*, otherwise it is *single-component*.

Our goal is to show that the total weight of bad contractions is bounded by an ε-fraction of the weight of F_0^*. We start by proving that if the set Q_t of terminals in C_t is sufficiently large, then the contraction is good. We define $\lambda := (1+\varepsilon)(p+c)/\varepsilon$.

▶ **Lemma 8.** *If $|Q_t| \geq \lambda$, then the contracted edge set C_t is good.*

Proof of Lemma 8. For brevity, we drop the index t. Let $r = w(C)/(|Q|-1)$ be the ratio of the contracted star, and let ℓ' be the number of deleted edges in D that connect two terminals. Note that any such edge has weight at least r, since it spans a star with two terminals, which has ratio equal to its weight, and since each edge in F^* (of which D is a subset) can only be heavier than the corresponding edge in the current graph G.

Let $u_1, \ldots, u_q$ be the Steiner vertices adjacent to edges in D, and let ℓ_i be the number of edges in D incident to one such Steiner vertex u_i. Consider the star spanned by the ℓ_i edges of D incident to u_i. If $\ell_i \geq 2$, the ratio of this star is at least r, since its edges are at least as heavy as the corresponding edges in G and the algorithm chose a star with the minimum ratio in G. Thus, the weight of edges in D incident to u_i is at least $r(\ell_i - 1)$. In the case where $\ell_i = 1$, the lower bound $r(\ell_i - 1) = 0$ on the weight holds trivially.

Any edge in D not incident to any Steiner vertex u_i connects two terminals. Therefore, we have $\ell' + \sum_{i=1}^{q} \ell_i = |D|$ as any edge in D is incident to a terminal in Q and we thus do not count any edge twice. Also recall that $|D| \geq |Q| - c$. Since F contains at most p Steiner vertices we have $q \leq p$, and we obtain

$$w(D) \geq r\ell' + \sum_{i=1}^{q} r(\ell_i - 1) = r\left(\ell' + \sum_{i=1}^{q} \ell_i - q\right) \geq r(|Q| - p - c)\,.$$

Finally, using $|Q| \geq \lambda$ we bound $w(C)$ by $(1+\varepsilon)w(D)$ as follows:

$$\begin{aligned}(1+\varepsilon)w(D) &\geq (1+\varepsilon)r(|Q| - p - c) = r(|Q| - 1) + r\big(\varepsilon|Q| - (1+\varepsilon)(p+c) + 1\big) \\ &\geq w(C) + r\left(\varepsilon\frac{(1+\varepsilon)(p+c)}{\varepsilon} - (1+\varepsilon)(p+c)\right) = w(C)\,.\end{aligned}$$

◀

Note that there may be a lot of contractions with $|Q| < \lambda$. However, we show that only a bounded number of them is actually bad. The key idea is to consider contractions with ratio in an interval $((1+\delta)^i; (1+\delta)^{i+1}]$ for some $\delta > 0$ and integer i. Due to the rescaling of weights every star belongs to an interval with $i \geq 0$. The following crucial lemma of our analysis shows that the number of bad single-component contractions in each such interval is bounded in terms of p and ε, if δ is a function of ε. In particular, let $\delta := \sqrt{1+\varepsilon} - 1$, so that $(1+\delta)^2 = 1 + \varepsilon$. We call an edge set C with ratio r in the i-th interval, i.e., with $r \in ((1+\delta)^i; (1+\delta)^{i+1}]$, an *$i$-contraction*, and define $\kappa := (1+\delta)p/\delta$.

▶ **Lemma 9.** *For any i the number of bad single-component i-contractions is at most κ.*

Proof. Let us focus on bad single-component i-contractions only which we just call bad i-contractions for brevity. Suppose for a contradiction that the number of bad i-contractions is larger than κ. Let $\tilde{t}$ be the first step with a bad i-contraction, i.e., $\tilde{t}$ is the minimum among all t for which $w(C_t) > (1+\varepsilon)w(D_t)$ and $w(C_t)/(|Q_t| - 1) \in ((1+\delta)^i; (1+\delta)^{i+1}]$. The plan is to show that at step $\tilde{t}$ there is a "light" star in $G_{\tilde{t}}$ with ratio at most $(1+\delta)^i$ and consequently the algorithm would do a j-contraction for some $j < i$. This leads to a contradiction, since we assumed that in step $\tilde{t}$ the contraction has ratio in interval i. Note that it is sufficient to find such a light star in $F^*_{\tilde{t}}$ as for each edge in $F^*_{\tilde{t}}$ there is an edge in the graph $G_{\tilde{t}}$ between the same vertices of the same weight or even lighter.

We claim that for each step t in which the algorithm does a bad i-contraction there is an edge $e_t \in D_t$ with weight at most $(1+\delta)^{i-1}$. We have $w(C_t) > (1+\varepsilon)w(D_t)$ as C_t is bad and $w(C_t) \leq (1+\delta)^{i+1}(|Q_t| - 1)$ as the ratio of C_t is in interval i. Putting it together and using the definition of δ we obtain $w(D_t) < (1+\delta)^{i+1}/(1+\varepsilon) \cdot (|Q_t| - 1) = (1+\delta)^{i-1}(|Q_t| - 1)$. Because C_t is single-component, we have $|D_t| \geq |Q_t| - 1$ and therefore there is an edge $e_t \in D_t$ with weight at most $(1+\delta)^{i-1}$, which proves the claim.

Since edges between terminals have weight at least $(1+\delta)^i$ in step t (recall that each such edge is a star with ratio equal to its weight), the edge e_t is incident to a Steiner vertex in F^*_t (otherwise the algorithm would have chosen one of the edges between terminals to contract). As $D_t \cap D_{t'} = \emptyset$, the edges e_t and $e_{t'}$ for steps $t \neq t'$ with bad i-contractions are distinct. Let S be the set of such light edges e_t. We have $|S| > \kappa$ as there are more than κ bad i-contractions.

Since the solution F^*_0 uses at most p Steiner vertices, also $F^*_{\tilde{t}}$ contains at most p of them. Therefore at $\tilde{t}$ there must be a Steiner vertex v in $F^*_{\tilde{t}}$ incident to at least $|S|/p > \kappa/p \geq (1+\delta)/\delta$ edges in S. Consider a star C with v as the center and with edges from S that are incident to v; we have $|C| \geq (1+\delta)/\delta$. The ratio of this star is at most $|C|(1+\delta)^{i-1}/(|C|-1)$.

Since $|C|/(|C|-1) \leq (1+\delta)$ (by a routine calculation) we get that the ratio of C is at most $(1+\delta)^i$ which is a contradiction to the assumption that the algorithm does an i-contraction in step $\tilde{t}$. ◀

We also need a bound on number of bad multiple-component edge sets.

▶ **Lemma 10.** *The number of steps t in which a bad multiple-component edge set C_t is contracted is at most $c-1$.*

Proof. If C_t is a bad multiple-component edge set, F^*_{t+1} must have at least one component fewer than F^*_t. Since F^*_0 has at most c components, the bound follows. ◀

We remark that the proofs of Lemmas 9 and 10 do not use that the number of terminals in a bad i-contraction is bounded by λ, as shown in Lemma 8. Instead we bound the total weight of bad contractions in terms of λ. For this let j be the largest interval of any contraction during the whole run of the algorithm, i.e., the ratio of every contracted star is at most $(1+\delta)^{j+1}$. As there are at most κ bad single-component contractions in each interval and c bad multiple-component contractions and the interval size grows exponentially, we can upper bound the total weight of bad contractions in terms of κ, c, λ and $(1+\delta)^j$. We can also lower bound the weight of $w(F^*_0)$ in terms of $(1+\delta)^j$ and the lower bound τ on the number of terminals in the graph. If τ is large enough then the total weight of edge sets C_t of bad contractions is at most $\varepsilon \cdot w(F^*_0)$. These ideas are summarized in the next lemma.

▶ **Lemma 11.** *Let j be the largest interval of any contraction during the whole run of the algorithm and let W_B be the total weight of edge sets C_t of bad contractions. Then, the following holds.*

1. $W_B \leq (\kappa+c)\cdot\lambda\cdot(1+\delta)^{j+2}/\delta$.
2. $w(F^*_0) \geq (1+\delta)^j \cdot (\tau - 2p - c)$.
3. *If* $\tau := (\kappa+c)\cdot\lambda\cdot(1+\delta)^2/(\varepsilon\delta) + 2p + c$ *then* $W_B \leq \varepsilon \cdot w(F^*_0)$.

The above lemma can now be used to prove that all the contractions put together (by scaling ε) form a $(1+\varepsilon)$-approximate pre-processing procedure with respect to F^*_0.

▶ **Lemma 12.** *The algorithm outputs an instance with $\tau \in \mathrm{O}\big((p+c)^2/\varepsilon^4\big)$ terminals and (together with the solution lifting algorithm) it is a $(1+2\varepsilon)$-approximate polynomial time pre-processing algorithm with respect to F^*_0.*

Note that in case the given p is smaller than the number of Steiner vertices in F^*_0, or c is smaller than the number of connected components in F^*_0, the algorithm still outputs a Steiner forest, but the approximation factor may be arbitrary. The last lemma can easily be used to prove Theorems 1, 4 and 5.

3 The unweighted directed Steiner tree problem

In this section we provide an EPAS for the Unweighted Directed Steiner Tree problem, in which each arc has unit weight. The idea behind our algorithm given in this section is to reduce the number of terminals of the input instance via a set of reduction rules. That is, we would like to reduce the input graph G to a graph H, and prove that the number of terminals in H is bounded by a function of our parameter p and the desired approximation ratio. On H we use the algorithm of Björklund et al. [2] to obtain an optimum solution.

Our first reduction rule represents the idea that a terminal in the immediate neighborhood of the root can be contracted to the root. Observe that in this case our algorithm has to

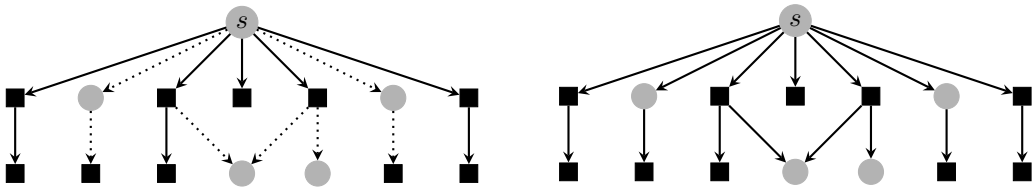


Figure 3 An example of extended neighborhood of Steiner vertex s. The set $N^0_{\mathrm{Ext}}(s)$ is depicted on the left using full arcs, while the vertices connected by dotted arcs are not a part of this set. The set $N^1_{\mathrm{Ext}}(s)$ is depicted on the right using full arcs.

pay 1 for connecting such a terminal to the root, however, any feasible solution must connect this terminal as well using at least one arc—this argument is formalized in Lemma 13.

▶ **Reduction Rule R1.** *If there is an arc from the root r to a terminal $v \in R$, we contract the arc (r, v), and declare the resulting vertex the new root.*

▶ **Lemma 13.** *Reduction Rule R1 is 1-safe and can be implemented in polynomial time. Furthermore, there is a solution lifting algorithm running in polynomial time and returning a Steiner arborescence if it gets a Steiner arborescence of the reduced graph as input.*

The idea behind our next reduction rule is the following. Assume there is a Steiner vertex s in the optimum arborescence T connected to many terminals with paths not containing any other Steiner vertices. We can then afford to buy all these paths emanating from s together with a path connecting the root to s. Formally, we say that a vertex u is a *k-extended neighbor* of some vertex v, if there exists a directed path P starting in v and ending in u, such that $V(P) \setminus \{v\}$ contains at most k Steiner vertices. Note that a vertex is always a k-extended neighbor of itself for any k, and that each of the above terminals connected to s in T is a 0-extended neighbor of s. We denote by $N^k_{\mathrm{Ext}}(v)$ the set of all k-extended neighbors of v, and call it the *k-extended neighborhood* of v (see Figure 3). By the following observation the Steiner vertex s of T lies in the p-extended neighborhood of the root r. Therefore there is a path containing at most p Steiner vertices connecting r to s.

▶ **Observation 14.** *Let $G = (V, A)$ be a directed graph with root $r \in R$. Suppose there exists a Steiner arborescence $T \subseteq G$ with at most p Steiner vertices. It follows that $V(T) \subseteq N^p_{Ext}(r)$.*

In what follows we fix $\varepsilon > 0$. The second reduction rule contracts a path from r to a Steiner vertex s in the p-extended neighborhood of r together with the 0-extended neighborhood of s if this neighborhood is sufficiently large.

▶ **Reduction Rule R2.** *If there exists a Steiner vertex s with $\left|N^0_{Ext}(s)\right| \geq p/\varepsilon$ and $s \in N^p_{Ext}(r)$, so that there is an $r \to s$ path P containing at most p Steiner vertices, then we contract the subgraph of G induced by $N^0_{Ext}(s)$ and P in G, and declare the resulting vertex the new root.*

▶ **Lemma 15.** *Reduction Rule R2 is $(1+\varepsilon)$-safe and can be implemented in polynomial time. Furthermore, there is a solution lifting algorithm running in polynomial time and returning a Steiner arborescence if it gets a Steiner arborescence of the reduced graph as input.*

Now we prove that if none of the above reduction rules is applicable and our algorithm was provided with a correct value for parameter p, then the number of terminals in the reduced graph can be bounded by p^2/ε.

▶ **Lemma 16.** *Let G be an instance of* Directed Steiner Tree*, and denote by H the graph obtained from G by exhaustive application of Reduction Rules R1 and R2. Suppose*

that there exists a Steiner arborescence in G containing at most p Steiner vertices. It follows that the remaining terminal set R of H has size less than p^2/ε.

The last step of the algorithm is to compute an optimum solution in the graph H obtained from the input graph G after exhaustively applying the two above reduction rules. From the resulting arborescence in H we obtain an arborescence in G by running the solution lifting algorithms for each reduction rule applied (in the reverse order); the existence and correctness of the solution lifting algorithms for our reduction rules is provided by Lemmas 13 and 15.

References

1. Ajit Agrawal, Philip Klein, and R. Ravi. When Trees Collide: An Approximation Algorithm for the Generalized Steiner Problem on Networks. *SIAM Journal on Computing*, 24(3):440–456, jun 1995. doi:10.1137/s0097539792236237.
2. Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74. ACM, 2007. doi:10.1145/1250790.1250801.
3. Edouard Bonnet, Bruno Escoffier, Eun Jung Kim, and Vangelis Th. Paschos. On subexponential and fpt-time inapproximability. In Gregory Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*, pages 54–65. Springer, 2013. doi:10.1007/978-3-319-03898-8_6.
4. Glencora Borradaile, Philip N. Klein, and Claire Mathieu. An $O(n \log n)$ approximation scheme for steiner tree in planar graphs. *ACM Trans. Algorithms*, 5(3):31:1–31:31, 2009. doi:10.1145/1541885.1541892.
5. Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6:1–6:33, 2013. doi:10.1145/2432622.2432628.
6. Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manuran gsi, Danupon Nanongkai, and Luca Trevisan. From Gap-ETH to FPT-Inapproximability: Clique, Dominating Set, and More. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, oct 2017. doi:10.1109/focs.2017.74.
7. Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. *J. Algorithms*, 33(1):73–91, 1999. doi:10.1006/jagm.1999.1042.
8. Yijia Chen and Bingkai Lin. The Constant Inapproximability of the Parameterized Dominating Set Problem. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, oct 2016. doi:10.1109/focs.2016.61.
9. Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi. Parameterized Approximation Algorithms for Directed Steiner Network Problems. *arXiv preprint*, 2017. arXiv:1707.06499.
10. Rajesh Hemant Chitnis, MohammadTaghi Hajiaghayi, and Guy Kortsarz. Fixed-parameter and approximation algorithms: A new look. In Gregory Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*, pages 110–122. Springer, 2013. doi:10.1007/978-3-319-03898-8_11.
11. Miroslav Chlebík and Janka Chlebíková. Approximation hardness of the steiner tree problem on graphs. In Martti Penttonen and Erik Meineche Schmidt, editors, *Algorithm Theory*

- *SWAT 2002, 8th Scandinavian Workshop on Algorithm Theory, Turku, Finland, July 3-5, 2002 Proceedings*, volume 2368 of *Lecture Notes in Computer Science*, pages 170–179. Springer, 2002. doi:10.1007/3-540-45471-3_18.

12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

13 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 624–633. ACM, 2014. doi:10.1145/2591796.2591884.

14 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and ids. *ACM Trans. Algorithms*, 11(2):13:1–13:20, 2014. doi:10.1145/2650261.

15 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

16 S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971. doi:10.1002/net.3230010302.

17 Pavel Dvořák, Andreas Emil Feldmann, Dušan Knop, Tomáš Masařík, Tomáš Toufar, and Pavel Veselý. Parameterized Approximation Schemes for Steiner Trees with Small Number of Steiner Vertices. *arXiv preprint*, 2017. arXiv:1710.00668v2.

18 Eduard Eiben, Mithilesh Kumar, Amer E Mouawad, and Fahad Panolan. Lossy kernels for connected dominating set on sparse graphs. *arXiv preprint*, 2017. arXiv:1706.09339.

19 David Eisenstat, Philip Klein, and Claire Mathieu. An efficient polynomial-time approximation scheme for Steiner forest in planar graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 626–638. Society for Industrial and Applied Mathematics, jan 2012. doi:10.1137/1.9781611973099.53.

20 Andreas Emil Feldmann. Fixed parameter approximations for k-center problems in low highway dimension graphs. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 588–600. Springer, 2015. doi:10.1007/978-3-662-47666-6_47.

21 Andreas Emil Feldmann and Dániel Marx. The complexity landscape of fixed-parameter directed steiner network problems. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.27.

22 Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 585–594. ACM, 2003. doi:10.1145/780542.780628.

23 Frank K Hwang, Dana S Richards, and Pawel Winter. *The Steiner tree problem*, volume 53. Elsevier, 1992. doi:10.1016/s0167-5060(08)x7008-6.

24 Mark Jones, Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Ondrej Suchý. Parameterized complexity of directed steiner tree on sparse graphs. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, volume 8125 of *Lecture Notes in Computer Science*, pages 671–682. Springer, 2013. doi:10.1007/978-3-642-40450-4_57.

25 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Plenum, 1972. doi:10.1007/978-1-4684-2001-2_9.

26 R. Krithika, Pranabendu Misra, Ashutosh Rai, and Prafullkumar Tale. Lossy kernels for graph contraction problems. In Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen, editors, *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India*, volume 65 of *LIPIcs*, pages 23:1–23:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.23.

27 Michael Lampis. Parameterized approximation schemes using graph widths. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 775–786. Springer, 2014. doi:10.1007/978-3-662-43948-7_64.

28 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 224–237. ACM, 2017. doi:10.1145/3055399.3055456.

29 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008. doi:10.1093/comjnl/bxm048.

30 Daniel Mölle, Stefan Richter, and Peter Rossmanith. Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. *Theory Comput. Syst.*, 43(2):234–253, 2008. doi:10.1007/s00224-007-9089-3.

31 Marcin Pilipczuk, Michal Pilipczuk, Piotr Sankowski, and Erik Jan van Leeuwen. Network Sparsification for Steiner Problems on Planar and Bounded-Genus Graphs. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*. IEEE, oct 2014. doi:10.1109/focs.2014.37.

32 Sebastian Siebertz. Lossy kernels for connected distance-r domination on nowhere dense graph classes. *arXiv preprint*, 2017. arXiv:1707.09819.

33 Ondrej Suchý. Extending the kernel for planar steiner tree to the number of steiner vertices. *Algorithmica*, 79(1):189–210, 2017. doi:10.1007/s00453-016-0249-1.

34 Andreas Wiese. A (1+epsilon)-approximation for unsplittable flow on a path in fixed-parameter running time. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 67:1–67:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.67.

35 David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011. doi:10.1017/cbo9780511921735.

Finding List Homomorphisms from Bounded-treewidth Graphs to Reflexive Graphs: a Complete Complexity Characterization

László Egri
Indiana State University, Department of Mathematics & Computer Science, Terre Haute, USA
Laszlo.Egri@indstate.edu

Dániel Marx
Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary
dmarx@sztaki.mta.hu

Paweł Rzążewski
Faculty of Mathematics and Information Science,
Warsaw University of Technology, Warsaw, Poland
p.rzazewski@mini.pw.edu.pl

Abstract

In the *list homomorphism* problem, the input consists of two graphs G and H, together with a list $L(v) \subseteq V(H)$ for every vertex $v \in V(G)$. The task is to find a homomorphism $\phi : V(G) \to V(H)$ respecting the lists, that is, we have that $\phi(v) \in L(v)$ for every $v \in V(H)$ and if u and v are adjacent in G, then $\phi(u)$ and $\phi(v)$ are adjacent in H. If H is a fixed graph, then the problem is denoted by LHOM(H). We consider the *reflexive* version of the problem, where we assume that every vertex in H has a self-loop. If is known that reflexive LHOM(H) is polynomial-time solvable if H is an interval graph and it is NP-complete otherwise [Feder and Hell, JCTB 1998].

We explore the complexity of the problem parameterized by the treewidth $\mathrm{tw}(G)$ of the input graph G. If a tree decomposition of G of width $\mathrm{tw}(G)$ is given in the input, then the problem can be solved in time $|V(H)|^{\mathrm{tw}(G)} \cdot n^{O(1)}$ by naive dynamic programming. Our main result completely reveals when and by exactly how much this naive algorithm can be improved. We introduce a simple combinatorial invariant $i^*(H)$, which is based on the existence of certain decompositions and incomparable sets, and show that this number should appear as the base of the exponent in the best possible running time. Specifically, we prove for every non-interval reflexive graph H that

- If a tree decomposition of width $\mathrm{tw}(G)$ is given in the input, then the problem can be solved in time $i^*(H)^{\mathrm{tw}(G)} \cdot n^{O(1)}$.
- Assuming the Strong Exponential-Time Hypothesis (SETH), the problem cannot be solved in time $(i^*(H) - \epsilon)^{\mathrm{tw}(G)} \cdot n^{O(1)}$ for any $\epsilon > 0$.

Thus by matching upper and lower bounds, our result exactly characterizes for every fixed H the complexity of reflexive LHOM(H) parameterized by treewidth.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness, Theory of computation → Graph algorithms analysis, Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Graph Homomorphism, List Homomorphism, Reflexive Graph, Treewidth

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.27

Funding This work was supported by ERC Starting Grant PARAMTIGHT (No. 280152), ERC Consolidator Grant SYSTEMATICGRAPH (No. 725978) of the European Research Council, and NSF EAGER Grant (No. 1751765).

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 27; pp. 27:1–27:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

It is well known that most NP-hard algorithmic graph problems can be solved significantly more efficiently on graphs of bounded treewidth than on general graphs. A large number of NP-hard problems are known to be fixed-parameter tractable (FPT) parameterized by treewidth, that is, if the input instance contains a tree decomposition of width w of the graph, then the problem can be solved in time $f(w) \cdot n^{O(1)}$ for some computable function f depending only on the width w. In recent years, there have been significant research efforts to understand how complexity depends on treewidth and to determine the best possible function $f(w)$ that can appear in the running time. On the algorithmic side, new algorithms with improved running times were obtained for a number of problems [6, 1, 25, 16]. On the complexity side, conditional lower bounds were given that, in many cases, match the running time of the best known algorithms, thereby giving a tight understanding of the complexity of the problem parameterized by treewidth [21, 20, 6, 22, 5]. These lower bounds are usually based on the Exponential-Time Hypothesis (ETH), which can be informally stated as n-variable 3-SAT cannot be solved in time $2^{o(n)}$, or on the Strong Exponential-Time Hypothesis (SETH), which can be informally stated as n-variable m-clause CNF-SAT cannot be solved in time $(2-\epsilon)^n \cdot m^{O(1)}$ for any $\epsilon > 0$.

As an exemplary result, let us consider the c-COLORING problem, where the task is to color the vertices of the graph with c colors such that adjacent vertices receive distinct colors. Using standard dynamic programming techniques, c-COLORING can be solved in time $c^{\mathrm{tw}(G)} \cdot n^{O(1)}$ if a tree decomposition of width $\mathrm{tw}(G)$ is given in the input. A result of Lokshtanov et al. [20] showed that this running time is essentially optimal.

▶ **Theorem 1** (Lokshtanov, Marx, and Saurabh [20]). *Let $c \geq 3$ be a fixed integer. Assuming the SETH, the c-COLORING problem on a graph G with n vertices, given with its tree decomposition of width $\mathrm{tw}(G)$, cannot be solved in time $\mathrm{poly}(n) \cdot (c-\epsilon)^{\mathrm{tw}(G)}$ for any $\epsilon > 0$.*

Homomorphisms. Given graphs G and H, a *homomorphism* from G to H is a mapping $\phi : V(G) \to V(H)$ such that if uv is an edge of G, then $\phi(u)\phi(v)$ is an edge of H. (In particular, if H has no loops, then this implies $\phi(u) \neq \phi(v)$ whenever u and v are adjacent.) For every fixed graph G, we can define the HOM(H) problem, where, given a graph G, the task is to find a homomorphism from G to H. Now c-COLORING is equivalent to HOM(K_c), where K_c is the clique on c vertices: it is easy to see that G is c-colorable if and only if it has a homomorphism to K_c. Thus the HOM(H) family of problems form a far-reaching generalization of the vertex coloring problem. A classic result of Hell and Nešetřil [17] characterized the complexity of HOM(H): it is polynomial-time solvable if H is bipartite and it is NP-complete for every nonbipartite H (see also [4, 18]).

What can we say about the complexity of HOM(H) parameterized by treewidth? It seems to be a natural goal to try to obtain, for every H, the best possible base c_H of the exponent that can appear in the running time $c_H^{\mathrm{tw}(G)} \cdot n^{O(1)}$. If H is the clique K_c, then we know from Theorem 1 that $c_H = c$, but what can we say about other graphs H? While this is a very natural question, it appears to be very difficult and deep as well: while the hardness of c-COLORING is well understood and can be easily exploited in hardness proofs such as Theorem 1, the hardness of HOM(H) for nonbipartite H comes from a somewhat mysterious combination of combinatorics and algebra [17, 4, 18, 23].

While we are unable at the moment to characterize the exact complexity of HOM(H) parameterized by treewidth, we resolve a related question that is still of interest, but apparently more tractable. The problem we study differs from the original question in two

ways. First, we are considering the list version of the problem: in an input instance of the *list homomorphism* LHOM(H) problem, each vertex v of G is equipped with a list $L(v) \subseteq V(H)$ and the task is to find a homomorphism ϕ from G to H that respects these lists, that is, $\phi(v) \in L(v)$ for every $v \in V(G)$. List versions of homomorphism and coloring problems are well studied [24, 7, 8, 13, 15, 12, 11, 14, 10, 9]. Typically, list versions are more robust than the ordinary versions and hardness proofs are simpler to prove for them. Feder et al. [10] characterized the polynomial-time solvable cases of LHOM(H): now it is not sufficient that H is bipartite, it has to be the complement of a circular arc graph, otherwise the problem is NP-complete. Second, we consider the *reflexive* version of the homomorphism problem, which means that we assume that every vertex of H has a self-loop attached to it. Thus even if u and v are adjacent in G, it is still possible that $\phi(u) = \phi(v)$ in a homomorphism ϕ from G to H. In particular, now there is always a homomorphism ϕ from every G to H: let us chose an arbitrary fixed vertex $u \in V(H)$ and let $\phi(v) = u$ for every $v \in V(G)$. However, it remains a nontrivial question whether there is a homomorphism from G to H that respects the lists $L(v)$ of the vertices of G. Feder and Hell [9] showed that reflexive LHOM(H) is polynomial-time solvable if H is an interval graph, and NP-complete otherwise. In general, the reflexive problem appears to have simpler structure and cleaner properties than the irreflexive version, where bipartiteness and parity issues introduce technical complications. We believe that it is reasonable to start with the reflexive problem as a prototype result.

Results. Our main result is exactly characterizing, for every fixed H, the complexity of reflexive list homomorphism parameterized by treewidth. Similarly to the c-COLORING problem, standard dynamic programming techniques give an algorithm with running time $|V(H)|^{\mathrm{tw}(G)} \cdot n^{O(1)}$ if a tree decomposition of width $\mathrm{tw}(G)$ is given (slightly more generally, we can also say that if every list $L(v)$ has size at most c, then the problem can be solved in time $c^{\mathrm{tw}(G)} \cdot n^{O(1)}$). However, unlike in the case of the c-COLORING problem, this algorithm is not necessarily optimal: for some H, we can actually do better. We identify two algorithmic ideas that can give improved algorithms:

- **Incomparable sets.** Suppose that the list $L(v)$ for some $v \in V(G)$ contains two vertices $a, b \in V(H)$, such that every neighbor of a (including a itself) is also a neighbor of b. It is easy to see that if there is a homomorphism ϕ from G to H with $\phi(v) = a$, then this can be modified to have $\phi(v) = b$ and it remains a valid homomorphism. In other words, vertex a in the list $L(v)$ is not necessary for the solution and can be removed from the list. Thus after a simple preprocessing step, we can assume that every $L(v)$ is an *incomparable set,* that is, $N[a] \subseteq N[b]$ does not hold for any two distinct $a, b \in L(v)$. This means that if we denote by $i(H)$ the maximum size of an incomparable set in H, then it can be assumed that every list has size at most $i(H)$ and hence the problem can be solved in time $i(H)^{\mathrm{tw}(G)} \cdot n^{O(1)}$. As $i(H)$ can be much less than $|V(H)|$, this running time can be significantly faster than $|V(H)|^{\mathrm{tw}(G)} \cdot n^{O(1)}$.
- **Decompositions.** We identify a certain kind of decomposition that can be used to simplify the problem. Formally, a decomposition is a partition (S, N, R) of the vertices of H such that $|S| \geq 2$, $|N \cup R| > 0$, N separates S and R, N induces a clique, and every vertex of S is adjacent to every vertex of N. As we show later, such a decomposition allows us to reduce LHOM(H) to instances of LHOM(H_1) and LHOM(H_2), where H_1 and H_2 are strict induced subgraphs of H.

We show that, in a formal sense, these two algorithmic ideas are sufficient to solve the problem as fast as possible. First, if the graph H is *undecomposable* (that is, does not have a decomposition as above), then the best possible running time is indeed of the form $i(H)^{\mathrm{tw}(G)} \cdot n^{O(1)}$. More generally, we define $i^*(H)$ to be the maximum of $i(H^*)$, taken over

every undecomposable, connected, non-interval induced subgraph H^* of H. Our main result shows that $i^*(H)^{\mathrm{tw}(G)} \cdot n^{O(1)}$ is the exact complexity of the problem.

▶ **Theorem 2.** *Let H be a connected reflexive non-interval graph with $k = i^*(H)$, and G be a graph with n vertices and treewidth* $\mathrm{tw}(G)$.

(a) *The* LHOM(H) *problem with instance (G, L) can be solved in time $poly(n + |H|) \cdot k^{\mathrm{tw}(G)}$ for any lists L, provided that G is given with its tree decomposition of width* $\mathrm{tw}(G)$.

(b) *There is no algorithm that solves* LHOM(H) *for every G and L in time $f(H) \cdot poly(n + |H|) \cdot (k - \epsilon)^{\mathrm{tw}(G)}$ for any computable function f and any $\epsilon > 0$, unless the SETH fails.*

Note that if H is a reflexive interval graph, then LHOM(H) is polynomial-time solvable [9] and if H is disconnected, then it is easy to reduce the problem to the components of H. Thus Theorem 2 gives a complete characterization of the complexity of the problem for every fixed H.

Let us discuss the significance of a complete classification result such as Theorem 2. As the LHOM(H) problem is an infinite family of problems, it is not clear at all what is the full range of algorithmic ideas that can help solve the problem faster than the naive $|V(H)|^{\mathrm{tw}(G)} \cdot n^{O(1)}$ time algorithm. Even after realizing that this naive algorithm can be beaten in some cases (e.g., by discovering the importance of incomparable sets or some form of decompositions), we cannot be sure that some completely different algorithm cannot solve some cases even faster, or can be applied to an even wider class of target graphs H. But in order to prove a complete classification result of the form of Theorem 2, one has to discover each and every relevant algorithmic idea. Our main result not only provides a set of algorithmic tools, but proves in a formal sense (assuming the SETH) that no other algorithmic idea can improve on these results. Thus we completely map the complexity landscape of the LHOM(H) problem, determining the complexity of every case of LHOM(H) with surprising tightness and revealing every combinatorial insight that can be exploited algorithmically.

Lower bound proofs. The complexity result of Theorem 2b needs to exploit three properties of the induced subgraph H^*: it is not an interval graph, it is undecomposable, and has a large incomparable set. There are well-known characterization results that show that every non-interval graph contains certain obstructions (induced cycles or asteroidal triples) and the NP-hardness proofs of Feder and Hell [9] show how these obstructions can be used to reduce 3-COLORING to LHOM(H). However, here we need something much stronger: if there is an incomparable set I of size $c = i^*(H)$, then we want to reduce c-COLORING to LHOM(H) and use the lower bound in Theorem 1. The natural idea is to represent the c colors of the c-COLORING problem by the c vertices appearing in the incomparable set I. Then the main challenge is to construct gadgets that express the $\neq$ relation, that is, ensure that two adjacent vertices are not assigned the same vertex of I, but every other combination is allowed. We show with a very delicate and technical proof that the incomparable set can be connected to the interval graph obstruction with a set of walks satisfying certain properties, and these walks, together with the obstruction, can be used to create the required gadgets. It turns out that, surprisingly, the only situation when we cannot find such walks is precisely when a decomposition exists. Thus if we assume that the graph is non-interval, has a large incomparable set, and has no decomposition, then we can construct the gadgets required for the reduction.

Exploiting decompositions. We finish the introduction with a brief explanation of how a decomposition (S, N, R) can be exploited (a more detailed algorithm description appears in

Section 3.1). As discussed above, we can assume that every $L(v)$ is an incomparable set in H. In particular, this means that if some $a \in S$ appears in $L(v)$, then $L(v)$ does not contain any vertex of N (as every vertex of N fully contains the neighborhood of every vertex in S). Let $X \subseteq V(G)$ be the set of vertices whose lists contain at least one vertex of S. The set X induces some number of connected components in G; let C be a connected component of $G[X]$.

The first crucial observation is that in every solution ϕ, one of the following two cases has to happen on C: either (1) $\phi(c) \in S$ for every $c \in C$, or (2) $\phi(c) \notin S$ for every $c \in C$. Otherwise, there would be two adjacent vertices $c_1, c_2 \in C$ with $\phi(c_1) \in S$ and $\phi(c_2) \notin S$. However, as N separates S and R in H, this is only possible if $\phi(c_2) \in N$, contradicting our earlier assumption. Thus as a first step, we check if there is a homomorphism ϕ_C from $G[C]$ to $H_1 := H[S]$ that respects the list. If there is no such homomorphism, then we can rule out the possibility that case (1) happens on C and remove the vertices of S from the lists of the vertices in C. Suppose now that there is such a homomorphism ϕ_C.

The second crucial observation is that if case (1) happens on C, then we might as well assume that the solution ϕ restricted to C is exactly the same as ϕ_C: it is easy to see that $\phi(v) \in N$ should hold for every $v \in N(C)$, and every vertex of N is adjacent to every vertex of S, hence no conflict can arise if we change ϕ to be the same as ϕ_C on C. Let us select an arbitrary vertex $a \in S$ and let us change the list of every $v \in C$ to be $L'(v) = (L(v) \setminus S) \cup \{a\}$, that is, the single vertex a will represent the vertices $L(v) \cap S$. We claim that this modification does not change the solvability of the instance: if the original instance has a solution where case (1) happens on C, then we can modify it to have a's on every vertex of C; and if we obtain a solution of the new instance with a's on C, then we can obtain a solution of the original instance by using ϕ_C on C.

We repeat these steps for every connected component C of $G[X]$. Then we obtain an instance where the selected vertex $a \in S$ is the only vertex of S that appears anywhere on the lists. This means that effectively we have an instance where we need to find a homomorphism to $H_2 := H \setminus (S \setminus \{a\})$. As $|S| \geq 2$, H_2 has strictly fewer vertices than H. Thus the existence of the decomposition (S, N, R) allowed us to reduce the problem to instances of $\text{LHOM}(H_1)$ and $\text{LHOM}(H_2)$ where H_1 and H_2 have fewer vertices than H.

2 Preliminaries

Throughout the paper we consider reflexive graphs only, i.e., we assume that for every vertex v, vv is an edge (a loop). Let $H = (V, E)$ be a reflexive graph. By $N[v]$ we denote the set $\{u \colon uv \in E\}$. Note that $v \in N[v]$. By $N(v)$ we denote $N[v] \setminus \{v\}$. For a set X of vertices, by $N[X]$ we denote $\bigcup_{v \in X} N[v]$, while $N(X)$ denotes $N[X] \setminus X$. For a set X and a vertex v, by $N_X[v]$ we denote $N[v] \cap X$. In an analogous way we define $N_X(v)$ and $N_X(Y)$ and $N_X[Y]$ for a set Y. For two disjoint sets $A, B \subseteq V$, such that no vertex from A is adjacent to a vertex from B, an A-B-separator is a set S, such that there is no path from any vertex $a \in A$ to any vertex $b \in B$ in the graph $H - S$. An A-B-separator S is *minimal* if no $S' \subsetneq S$ is an A-B-separator. If A is a singleton, say $A = \{a\}$, we write a-B-separator instead of $\{a\}$-B separator (analogously if B is a singleton).

For two graphs G and H, a mapping $f \colon V(G) \to V(H)$ is a *homomorphism* if for every edge xy of G it holds that $f(x)f(y)$ is an edge of H. If f is a homomorphism from G to H, we denote it shortly by $f \colon G \to H$. We write $G \to H$ to say that there exists some homomorphism from G to H. For a fixed graph H, by $\text{HOM}(H)$ we consider an algorithmic problem of deciding if there is a homomorphism from a given graph G to H.

In the *list homomorphism* problem we are given two graphs G, H and a mapping $L\colon V(G) \to 2^{V(H)}$ (where the sets $L(v)$ for $v \in V(G)$ are called *H-lists* or just *lists*), and we ask for a homomorphism $f\colon G \to H$, such that $f(x) \in L(x)$ for every $x \in V(G)$. We denote this by $f\colon (G, L) \to H$. We often write $(G, L) \to H$ to denote that there is a list homomorphisms from G to H with lists L. Moreover, as we only deal with list homomorphisms, we write $f\colon G \to H$ to denote $f\colon (G, L) \to H$, if the lists are clear from the context. For a fixed graph H, by LHOM(H) we denote the algorithmic problem, whose input is a graph G with lists L, and we ask whether there exists a list homomorphism $(G, L) \to H$.

We observe that if H has several connected components, then there is a polynomial-time reduction from LHOM(H) to the problems LHOM(H') for the connected components H' of H. Thus we always assume that H is connected.

2.1 Interval graphs and obstructions

Interval graphs are one of the most studied classes of geometric intersection graphs. A graph H is an interval graph if it admits an *interval representation*, where each vertex is represented by some closed interval of the real line and two vertices are adjacent if and only if their corresponding intervals intersect. Note that interval graphs are usually defined to be irreflexive, but in our case we consider reflexive graphs.

Before we analyze structural properties of interval graphs, we need a few more definitions. An *asteroidal triple* is an independent set of three vertices a, b, c, such that for every $\{i, j, \ell\} = \{a, b, c\}$ there is an i-j-path $\mathcal{W}_{i,j}$, whose every vertex is non-adjacent to ℓ. Note that by our convention $\mathcal{W}_{j,i}$ is $\mathcal{W}_{i,j}$ reversed.

▶ **Theorem 3** (Lekkeikerker and Boland [19]). *A graph is an interval graph if and only if does not contain an asteroidal triple or an induced cycle of length at least 4.*

There is a deep connection between the list homomorphism problem and reflexive interval graphs, as shown in the following dichotomy theorem of Feder and Hell [9].

▶ **Theorem 4** (Feder and Hell [9]). *Let H be a reflexive graph. If H is an interval graph, then the* LHOM(H) *problem is polynomially solvable, otherwise it is NP-complete.*

In this paper we will focus on graphs H, for which LHOM(H) is NP-complete, so we will assume that H is non-interval and thus contains at least one of structures mentioned in Theorem 3.

Observe that for an asteroidal triple a, b, c, we may w.l.o.g. assume that each path $\mathcal{W}_{a,b}$ is induced. We define the *asteroidal subgraph* of an asteroidal triple a, b, c, as the subgraph of H induced by $\mathcal{W}_{a,b} \cup \mathcal{W}_{b,c} \cup \mathcal{W}_{a,c}$.[1] For a vertex a (b, c, resp.), we say that the path $\mathcal{W}_{b,c}$ ($\mathcal{W}_{a,c}$, $\mathcal{W}_{a,b}$, resp.) is *opposite*.

Moreover, note that an induced cycle with at least 6 vertices contains an asteroidal subgraph. So an equivalent statement of Theorem 3 says that every non-interval graph H contains an induced 4-cycle, an induced 5-cycle, or an asteroidal subgraph. An induced subgraph of H isomorphic to one of these three structures is called an *obstruction* in H.

A vertex $o \in \mathbb{O}$ is a *corner* if:

- $\mathbb{O}$ is isomorphic to a 4-cycle or a 5-cycle, or
- $\mathbb{O}$ is an asteroidal subgraph for an asteroidal triple containing o.

Two vertices o, o' of an obstruction $\mathbb{O}$ are *opposite* if:

[1] We will often identify graphs with their vertex sets.

- both are corners, or
- $\mathbb{O}$ is an asteroidal subgraph and o belongs to the path opposite to o'.

2.2 Dominating vertices and incomparable sets

For two vertices u, v of H, we say that v *dominates* u or, equivalently, u is *dominated* by v, if $N[u] \subseteq N(v)$. Observe that this implies that u and v are adjacent. We say that a set X *dominates* a set Y if every $x \in X$ dominates every $y \in Y$. If u is not dominated by v, it means that there is a vertex $u' \in N[u]$ (possibly $u' = u$), which is not a neighbor of v. Two vertices u and v are *incomparable* if u does not dominate v and v does not dominate u. A set S of vertices is *incomparable* if all its members are pairwise incomparable. By $i(H)$ we denote the size of the largest incomparable set in H.

2.3 Avoiding walks

A *walk* is a sequence of vertices $\mathcal{P} = p_1, p_2, \ldots, p_\ell$, such that $p_i p_{i+1}$ is an edge for every $i = 1, 2, \ldots, \ell - 1$. For the walk $\mathcal{P}$, its *length* denotes the number $\ell - 1$. For two vertices a, b, we say that $\mathcal{P} = p_1, p_2, \ldots, p_\ell$ is an a-b-walk if $p_1 = a$ and $p_\ell = b$. We denote this shortly by $\mathcal{P} \colon a \to b$. By $\bar{\mathcal{P}}$ we denote the *reversed walk*, i.e., $\bar{\mathcal{P}} = p_\ell, p_{\ell-1}, \ldots, p_2, p_1$.

For two walks $\mathcal{A} = a_1, a_2, \ldots, a_\ell$ and $\mathcal{B} = b_1, b_2, \ldots, b_{\ell'}$ such that $a_\ell = b_1$, we let $\mathcal{A} \circ \mathcal{B}$ denote the *concatenation* of $\mathcal{A}$ and $\mathcal{B}$, i.e., the walk $a_1, a_2, \ldots, a_\ell, b_2, b_3, \ldots, b_{\ell'}$. Note that $|\mathcal{A} \circ \mathcal{B}| = |\mathcal{A}| + |\mathcal{B}| - 1$.

For two walks $\mathcal{P} = p_1, p_2, \ldots, p_\ell$ and $\mathcal{Q} = q_1, q_2, \ldots q_\ell$ of equal length, we say that $\mathcal{P}$ *avoids* $\mathcal{Q}$ if p_i is non-adjacent to q_{i+1} for every $i = 1, 2, \ldots, \ell - 1$. We conclude this section with two simple observations concerning walks and avoidance.

▶ **Observation 5.** *For walks $\mathcal{A} \colon a \to b$, $\mathcal{B} \colon b \to c$ and $\mathcal{A}' \colon a' \to b', \mathcal{B}' \colon b' \to c'$, if $\mathcal{A}$ avoids $\mathcal{A}'$ and $\mathcal{B}$ avoids $\mathcal{B}'$, then $\mathcal{A} \circ \mathcal{B}$ avoids $\mathcal{A}' \circ \mathcal{B}'$.* ◀

▶ **Observation 6.** *Let $\mathcal{P} = p_1, p_2, \ldots, p_\ell$ and $\mathcal{Q} = q_1, q_2, \ldots q_\ell$ be two walks, such that $\mathcal{P}$ avoids $\mathcal{Q}$. Then $\bar{\mathcal{Q}}$ avoids $\bar{\mathcal{P}}$.* ◀

3 Algorithm

In this section we prove the algorithmic part of our main result, i.e., Theorem 2a). Let us start with the following simple observation.

▶ **Observation 7.** *Let u, v be vertices of H, such that v dominates u. Let $f \colon G \to H$ be a homomorphism, such that $f(x) = u$ for some vertex x of G. Then f' defined by $f'(x) := v$ and $f'(y) := f(y)$ for every $y \in V(G) \setminus \{x\}$ is also a homomorphism from G to H.* ◀

Thus we can assume that in our instance (G, L) of LHOM(H) the set $L(x)$ is incomparable for every vertex x of G (otherwise we can safely remove a dominated vertex).

3.1 Decomposition

For a graph H, let $T(H, n, t)$ denote an upper bound for the time complexity of an algorithm solving the LHOM(H) problem on a graph with n vertices and treewidth t. The following lemma is the main tool in the proof of Theorem 2a). The proof is omitted in this extended abstract.

▶ **Lemma 8** (Decomposition lemma). *Let $H = (V, E)$ be a reflexive graph, whose vertex set can be partitioned into three subsets S, N, R, such that:*

1. $|S| \geq 2$,
2. *N is a clique with at least one vertex,*
3. *N separates S and R,*
4. *all edges between S and N are present in H.*

Let H_1 be the subgraph of H induced by S, and H_2 be the subgraph of H obtained by contracting S to a single vertex. Moreover, suppose that there are constants c, d, such that $T(H_1, n, t) = O(c^t \cdot n^d)$ and $T(H_2, n, t) = O(c^t \cdot n^d)$. Then $T(H, n, t) = O(c^t \cdot n^d)$.

A graph H which satisfies the assumptions of Lemma 8 is called *decomposable* and we say that (S, N, R) is a *decomposition* of H, or that H *decomposes* into H_1 and H_2. We refer to S as *dominated part* and the set N as *dominating clique separator*. A graph which is not decomposable is called *undecomposable*.

Observe that with H we can associate a *decomposition tree* $\mathcal{T}$, whose nodes are labeled with induced subgraphs of H. The root, denoted by $node(H)$ corresponds to the whole graph H. If H is undecomposable, then the decomposition tree has just one node. If H decomposes into H_1 and H_2, then $node(H)$ has two children, $node(H_1)$ and $node(H_2)$, respectively. We construct a decomposition tree recursively. Clearly, each leaf of the decomposition tree is an undecomposable induced subgraph of H. Note that a decomposition tree may not be unique, as a graph may have more than one decomposition. However, the number of leaves is always $O(|H|)$, so the total number of nodes is also $O(|H|)$.

3.2 Solving LHom(H) problem

Now we are ready to present an algorithm for determining if $(G, L) \to H$.

Proof of Theorem 2a). We assume that the graph G has n vertices and is given along with its tree decomposition of width $\mathrm{tw}(G)$. We also define

$$i^*(H) := \max\{i(H') \colon H' \text{ is undecomposable connected non-interval induced subgraph of } H\}.$$

Observe that if H' is an induced subgraph of H, then $i^*(H') \leq i^*(H)$, and thus $i(H) = i^*(H)$ for undecomposable H.

It can be shown that in time polynomial in H we can check if H is undecomposable, or find a decomposition. If H is undecomposable, we run a standard dynamic programming on a tree decomposition of G (see [3, 2]). For each bag of the tree decomposition we store all partial list homomorphisms from the graph induced by this bag to H. By Observation 7, the size of each list $L(x)$ for $x \in V(G)$ is at most $i(H)$, thus the complexity of the dynamic programming algorithm is bounded by $O(n^d \cdot i(H)^{\mathrm{tw}(G)}) = O(n^d \cdot i^*(H)^{\mathrm{tw}(G)})$ for some constant d.

So suppose H is decomposable. Let $\mathcal{T}$ be a decomposition tree of H, note that it can be constructed in polynomial time, has $O(|H|)$ nodes, and its every leaf corresponds to an induced subgraph of H with strictly fewer vertices. Indeed, if H decomposes into H_1 and H_2, then they are both induced subgraphs of H and $|H_1|, |H_2| < |H|$. Therefore, for any leaf H' of $\mathcal{T}$, we can solve every instance of LHom(H') with n vertices and treewidth at most $\mathrm{tw}(G)$ in time $O(n^d \cdot i^*(H)^{\mathrm{tw}(G)})$ (note that this is also true if H' is an interval graph, as then we can use a polynomial algorithm). Now, applying Lemma 8 in a bottom-up fashion, we conclude that we can solve LHom(H) in time $O(n^d \cdot i^*(H)^{\mathrm{tw}(G)})$, which completes the proof of Theorem 2a). ◀

4 Hardness

In this section we prove Theorem 2b), i.e., the lower bound for an algorithm deciding the existence of a list homomorphism $(G, L) \to H$. We will prove the following theorem.

▶ **Theorem 9.** *Let H be a connected, reflexive, undecomposable graph with $i(H) \geq 3$. Assuming the SETH, there is no algorithm that solves* LHom(H) *for every G and L in time $f(H) \cdot poly(|G| + |H|) \cdot (i(H) - \epsilon)^{\mathrm{tw}(G)}$ for any $\epsilon > 0$ and any computable function f.*

Let us first show that Theorem 9 is equivalent to Theorem 2b).

Theorem 9 → Theorem 2b). Suppose Theorem 9 holds and Theorem 2b) fails. So there is a graph H (may be decomposable) and an algorithm A that solves LHom(H) in time $f(H) \cdot poly(|G| + |H|) \cdot (i^*(H) - \epsilon)^{\mathrm{tw}(G)}$ for every input G, L. Let H' be an undecomposable connected non-interval induced subgraph of H, such that $i(H') = i^*(H)$. As every instance of LHom(H') can be seen as an instance of LHom(H), the algorithm A can be used to solve LHom(H') in time $f'(H') \cdot poly(|G| + |H'|) \cdot (i(H') - \epsilon)^{\mathrm{tw}(G)}$, thus contradicting Theorem 9.

Theorem 2b) → Theorem 9. Suppose Theorem 2b) holds and Theorem 9 fails. So there is an undecomposable graph H and an algorithm A that solves LHom(H) in time $f(H) \cdot poly(|G| + |H|) \cdot (i(H) - \epsilon)^{\mathrm{tw}(G)}$ for every input G, L. But since H is undecomposable, we have $i^*(H) = i(H)$, so algorithm A contradicts Theorem 2b).

4.1 Using an obstruction to express basic relations

Let $\mathbb{O}$ be an obstruction in H with non-adjacent corners α, β and let $k \geq 2$ be an integer. First, we show how express k-wise relations $OR_k = \{\alpha, \beta\}^k \setminus \alpha^k$ and $NAND_k = \{\alpha, \beta\}^k \setminus \beta^k$. More formally, we define a graph $F(OR_k)$ ($F(NAND_k)$, resp.), called an *OR_k-gadget* (*$NAND_k$-gadget*, resp.) with H-lists L and k specified vertices $x_1, x_2, \ldots, x_k$, such that:

- for every $i \in [k]$ it holds that $L(x_i) = \{\alpha, \beta\}$,
- the relation $\bigcup_{f\colon F(OR_k) \to H} \{f(x_1)f(x_2)\ldots f(x_k)\}$ is exactly OR_k (respectively, $\bigcup_{f\colon F(NAND_k) \to H} \{f(x_1)f(x_2)\ldots f(x_k)\}$ is $NAND_k$).

The construction of these gadgets is simple and it is omitted in this extended abstract. Another useful property of obstructions is shown in the following lemma.

▶ **Lemma 10** (Moving inside the obstruction)**.** *Let $\mathbb{O}$ be an obstruction with distinct corners a, c. Moreover, let b, d be distinct vertices of $\mathbb{O}$, such that b is a corner and d is either a corner, or a vertex non-adjacent to b. Then there are walks $\mathcal{A}_{a,b}, \mathcal{A}'_{a,b} : a \to b$ and $\mathcal{B}_{c,d}, \mathcal{B}'_{c,d} : c \to d$, such that $\mathcal{A}_{a,b}$ avoids $\mathcal{B}_{c,d}$ and $\mathcal{B}'_{c,d}$ avoids $\mathcal{A}'_{a,b}$. Moreover, all four walks use only vertices of $\mathbb{O}$ and can be constructed in polynomial time.*

Proof. If $\mathbb{O}$ is an induced 4-cycle or an induced 5-cycle, the walks are easy to construct. So consider the case that $\mathbb{O}$ is an asteroidal subgraph for an asteroidal triple o_1, o_2, o_3.

Case 1. First, let us deal with case when both b, d are corners. Then we have $\{a, b, c, d\} \subseteq \{o_1, o_2, o_3\}$. If $a = b$ and $c = d$ then the problem is trivial. If $a = b$ and $c \neq d$, then we set $\mathcal{A}_{a,b} = \mathcal{A}'_{a,b} = a, a, \ldots, a$ and $\mathcal{B}_{c,d} = \mathcal{B}'_{c,d} = \mathcal{W}_{c,d}$ (the walk opposite to a). The case when $a \neq b$ and $c = d$ is similar. So we assume that $a \neq b$ and $c \neq d$. If $a = d$ and $c \neq b$ (the case when $a \neq d$ and $c = b$ is similar), we set $\mathcal{A}_{a,b} = \mathcal{A}'_{a,b} = \mathcal{W}_{a,b} \circ b, b, \ldots, b$ and $\mathcal{B}_{c,d} = \mathcal{B}'_{c,d} = c, c, \ldots, c \circ \mathcal{W}_{c,a}$. If $a = d$ and $c = b$, we set $\mathcal{A}_{a,b} = \mathcal{A}'_{a,b} = a, a, \ldots, a \circ \mathcal{W}_{a,c} \circ c, c, \ldots, c$ and $\mathcal{B}_{c,d} = \mathcal{B}'_{c,d} = \mathcal{W}_{c,b} \circ b, b, \ldots, b \circ \mathcal{W}_{b,a}$.

Case 2. Next, consider the case when b is a corner and d is not. We know that $d \in \mathcal{W}_{b',b''}$, where b', b'' are corners and $b' \neq b$. Let $\mathcal{W}'$ be the subpath of $\mathcal{W}_{b',b''}$, starting in b' and ending in d. Recall that $\mathcal{W}_{b',b''}$ is induced, so even if $b = b''$, there is no edge from $\mathcal{W}'$ to b. We set

$$\begin{array}{llllllll} \mathcal{A}_{a,b} = & \mathcal{C}_{a,b} & \circ & b, b, \ldots, b & \mathcal{A}'_{a,b} = & \mathcal{C}'_{a,b} & \circ & b, b, \ldots, b \\ \mathcal{B}_{c,d} = & \mathcal{D}_{c,b'} & \circ & \mathcal{W}' & \mathcal{B}'_{c,d} = & \mathcal{D}'_{c,b'} & \circ & \mathcal{W}', \end{array}$$

where $\mathcal{C}_{a,b}, \mathcal{C}'_{a,b} \colon a \to b$ and $\mathcal{D}_{c,b'}, \mathcal{D}'_{c,b'} \colon c \to b'$ are appropriate walks given by Case 1. ◀

4.2 Constructing distinguishing walks

As we have seen, we can easily use the structure of an obstruction to enforce non-trivial relations that could be used to show hardness. For the rest of the proof we will show that we can attach vertices of an incomparable set to the vertices of an obstruction using walks with certain avoidance properties, which will later be exploited to prove hardness.

For a walk $\mathcal{P} = v_1, v_2, \ldots, v_n$, by $\widetilde{\mathcal{P}}$ we denote the walk $\mathcal{P}$ with its first vertex removed, i.e., $\widetilde{\mathcal{P}} = v_2, \ldots, v_n$. The following structural lemmas will be later used to obtain the main gadget used in our hardness proof.

▶ **Lemma 11.** *Let H be a connected undecomposable non-interval reflexive graph, and $\mathbb{O}$ be an obstruction in H with non-adjacent corners α, β. Let $\mathcal{S}$ be a set of incomparable vertices in H such that $|\mathcal{S}| \geq 2$. Let a and b be arbitrary distinct vertices in $\mathcal{S}$. Then there is a partition (X, Y) of $\mathcal{S}$ such that vertices a and b are in X, and there are walks $\mathcal{D}_v$ for each $v \in \mathcal{S}$ of length at least 1, satisfying the following properties:*

1. *For each $v \in \mathcal{S}$, the first vertex of $\mathcal{D}_v$ is v, and its last vertex is either α ($\mathcal{D}_v$ is said to be an α-walk) or β ($\mathcal{D}_v$ is said to be a β-walk).*
2. *$\mathcal{D}_a$ is an α-walk and $\mathcal{D}_b$ is a β-walk.*
3. *Let $u, v \in \mathcal{S}$ such that $\mathcal{D}_u$ is an α-walk and $\mathcal{D}_v$ is a β-walk. Then*
   a. *if $u, v \in X$, or if $u \in Y$ and $v \in X \cup Y$, then $\mathcal{D}_u$ avoids $\mathcal{D}_v$,*
   b. *if $u \in X$, $v \in Y$, then $\widetilde{\mathcal{D}}_u$ avoids $\widetilde{\mathcal{D}}_v$.*
4. *For any $v \in Y$ and $u \in X$, there is no edge joining v and the second vertex of $\mathcal{D}_u$.*
5. *For every $v \in Y$, the second vertex of $\mathcal{D}_v$ is in Y.*

Recall that it is possible that we have two walks $\mathcal{D}_x, \mathcal{D}_y$ constructed in Lemma 11, such that $\mathcal{D}_x$ is an α-walk, $\mathcal{D}_y$ is a β-walk, but $\mathcal{D}_x$ **does not** avoid $\mathcal{D}_y$ (this may happen for $x \in X$ and $y \in Y$). This is an undesired situation for us, but luckily such walks have a well-defined structure. In the next lemma we will construct a small gadget to patch this situation, and then we will combine them to construct the main tool in our hardness proof, i.e., a *distinguisher gadget*.

▶ **Lemma 12.** *Let $H, \mathcal{S}, X, Y$, and $\mathcal{D}_v$, where $v \in \mathcal{S}$, be as in Lemma 11. Let $N_X = \{d_2^x \colon x \in X\}$, i.e., the set of vertices that appear as a second vertex of a walk $\mathcal{D}_x$ where $x \in X$ and $N_Y = \{d_2^y \colon y \in Y\}$. Then there is a graph F with H-lists and two specified vertices $p_1, p_2 \in V(F)$ such that*

1. *$L(p_1) = \mathcal{S}$ and $L(p_2) = N_X \cup N_Y$,*
2. *for any list homomorphism $\varphi \colon F \to H$, if $\varphi(p_1) \in X$, then $\varphi(p_2) \notin Y$,*
3. *for every $v \in \mathcal{S}$, there is $\psi \colon F \to H$, such that $\psi(p_1) = v$ and $\psi(p_2) = d_2^v$.*

4.3 Constructing a distinguisher gadget

The main tool in our hardness proof is a gadget called a *distinguisher*. Let H be an undecomposable reflexive non-interval graph with obstruction $\mathbb{O}$ with non-adjacent corners α, β. For an incomparable set $\mathcal{S}$ of H and two vertices $a, b \in \mathcal{S}$, a distinguisher is a graph $D_{a/b}$ with two specified vertices x, y and H-lists L, such that:

1. $L(x) = \mathcal{S}$ and $L(y) = \{\alpha, \beta\}$,
2. there is a list homomorphism $\phi_a \colon D_{a/b} \to H$, such that $\phi_a(x) = a$ and $\phi_a(y) = \alpha$,
3. there is a list homomorphism $\phi_b \colon D_{a/b} \to H$, such that $\phi_b(x) = b$ and $\phi_b(y) = \beta$,
4. for any $c \in \mathcal{S} \setminus \{a, b\}$ there is $\phi_c \colon D_{a/b} \to H$, such that $\phi_c(x) = c$ and $\phi_c(y) \in \{\alpha, \beta\}$,
5. there is no list homomorphism $\phi \colon D_{a/b} \to H$, such that $\phi(x) = a$ and $\phi(y) = \beta$.

▶ **Lemma 13** (Construction of distinguisher). *Let $H = (V, E)$ be an undecomposable reflexive non-interval graph with obstruction $\mathbb{O}$ with two non-adjacent corners α, β. Let $\mathcal{S}$ be a maximum incomparable set in H. Then for every ordered pair (a, b) of distinct elements of $\mathcal{S}$ there exists a distinguisher $D_{a/b}$.*

Proof. Call Lemma 11 for $H, \mathcal{S}, a, b$ to obtain a partition (X, Y) of $\mathcal{S}$ and walks $\mathcal{D}_v$ for every $v \in \mathcal{S}$. Let s be the length of each of these walks. By d_j^v we denote the j-th vertex of $\mathcal{D}_v$.

Let P be a path with s vertices $p_1, p_2, \ldots, p_s$. We set $L(p_j) = \bigcup_{v \in \mathcal{S}} \{d_j^v\}$. Observe that by Lemma 11 we have $s \geq 2$.

Next, call Lemma 12 to obtain a graph F_P and unify its p_1-vertex with p_1 of P and its p_2-vertex with p_2 of P. Observe that this unification preserves lists. Finally, we set $x = p_1$ and $y = p_s$. Let us verify that the graph constructed in such a way is indeed a distinguisher. The first property holds by the definition of the walks $\mathcal{D}_v$ for $v \in \mathcal{S}$. To show properties 2,3, and 4, consider $v \in \mathcal{S}$ and set $\phi_v(p_i) = d_i^v$ for all $i \in [s]$. This mapping can be extended to the vertices of F_P by property 3 of Lemma 12.

Finally, let us show that property 5 holds as well. Assume for the sake of contradiction that a list homomorphism $\phi \colon D_{a/b} \to H$, such that $\phi(x) = a$ and $\phi(y) = \beta$, exists. Observe that $\phi(p_1), \phi(p_2), \ldots, \phi(p_s)$ is an a-β walk of length s in H, such that for every $i \in [s]$ we have $\phi(p_i) \in \bigcup_{v \in \mathcal{S}} \{d_2^v\}$. For all $i \in [s]$, let $\mathcal{D}^i$ denote a walk from $\{\mathcal{D}_v \colon v \in \mathcal{S}\}$, whose i-th vertex is $\phi(p_i)$ (if there is more than one such a walk, choose an arbitrary one). Observe that $\mathcal{D}^1 = \mathcal{D}_a$ is an α-walk. Let i be a minimum integer, such that $\mathcal{D}^i$ is a β-walk. This value is well-defined, as $\mathcal{D}^s$ is a β-walk. Thus there is an edge between the $(i-1)$th vertex of the α-walk $\mathcal{D}^{i-1}$ and the i-th vertex of the β-walk $\mathcal{D}^i$, so $\mathcal{D}^{i-1}$ does no avoid $\mathcal{D}_i$. Let u, v be vertices such that $\mathcal{D}^{i-1} = \mathcal{D}_u$ and $\mathcal{D}^i = \mathcal{D}_v$. If $u, v \in X$, or $u \in Y$ and $v \in X \cup Y$, then we have a contradiction with property 3a in Lemma 11.

Thus assume that $u \in X$ and $v \in Y$. If $i \geq 3$, then again we have a contradiction with property 3b in Lemma 11. Thus the only case left is $i = 2$. By property 2. in Lemma 12 we observe that $d_2^v \notin Y$, and thus, by property 5 in Lemma 11, we conclude that $v \notin Y$, a contradiction. This completes the proof. ◀

4.4 Hardness proof

We are ready to prove to prove Theorem 9. Recall that Theorem 9 implies Theorem 2b).

Proof of Theorem 9. Let $\mathbb{O}$ be an obstruction in H and let α, β be non-adjacent corners of $\mathbb{O}$. Let $\mathcal{S} = \{v_1, v_2, \ldots, v_k\}$ be a maximum incomparable set in H. Note that we can assume that $k \geq 3$, since the corners of $\mathbb{O}$ are pairwise incomparable.

Suppose we are given a graph G along with its tree decomposition of width $\mathrm{tw}(G)$. The main idea of our hardness proof is to construct a graph G^* with H-lists L such that:

- $(G^*, L) \to H$ if and only if G is k-colorable (in our construction the colors used on G will correspond to vertices from $\mathcal{S}$),
- the number of vertices of G^* is $g(H) \cdot (|V(G)| + |E(G)|)$ for some function g of H,
- the treewidth of G^* is at most $g(H) + \mathrm{tw}(G)$,
- G^* can be constructed in time $poly(|V(G)|) \cdot g'(H)$ for some function g'.

Invoking Theorem 1, this will prove Theorem 9. The construction is performed in four steps.

Step 1. Constructing an indicator gadget. Fix $i \in [k]$. For $j \in [k] \setminus \{i\}$, we use Lemma 13 to construct a distinguisher gadget D_{v_i/v_j} with two specified vertices $x_{i,j}$ and $y_{i,j}$. The number of constructed gadgets is thus $k-1$. We identify vertices $x_{i,j}$ for all $j \in [k]$, let us call this identified vertex x_i. Moreover, introduce a new vertex c_i. Now, using the construction from Section 4.1 we introduce an OR_k gadget and identify its specified vertices with distinct vertices from $X_i := \{c_i\} \cup \bigcup_{j \in [k]\setminus\{i\}} \{v_{i,j}\}$ (there are k vertices in this set). Let us call this graph I_i ('I' stands for *indicator*).

The construction forces that in every list homomorphism $f\colon I_i \to H$, at least one vertex from X_i is mapped to β. Observe that:

- for every $f\colon I_i \to H$, if $f(x_i) = v_i$, then $f(c_i) = \beta$.
- for every $j \neq i$, there exist $f', f''\colon I_i \to H$, such that $f'(x_i) = f''(x_i) = v_j$ and $f'(c_i) = \alpha$ and $f''(c_i) = \beta$.

Step 2. Constructing a half-edge gadget. Let us construct k indicator gadgets $I_1, I_2, \ldots, I_k$. We identify the vertices $x_1, x_2, \ldots, x_k$, and call this vertex x. Call the resulting gadget a *half-edge*. By the construction of indicators, we observe that for a half-edge F, the following hold:

- for every $f\colon F \to H$, if $f(x) = v_i$, then $f(c_i) = \beta$,
- for every $i \in [k]$, and every tuple $X \in \{\alpha, \beta\}^k$, such that $X_i = \beta$, there exists $f\colon F \to H$ such that $f(x) = v_i$ and $f(c_j) = X_j$.

Step 3. Constructing an edge gadget. An *edge gadget* consists of two half-edge gadgets F, F' (we will use primes to denote the vertices in F'). Moreover, for every $i \in [k]$, we introduce a $NAND_2$ gadget on vertices c_i and c'_i, which enforces that at least one of them is mapped to α. We call the resulting graph an *edge gadget*. For an edge gadget FF we observe the following:

- for any $f\colon FF \to H$, if $f(x) = v_i$, then $f(x') \neq v_i$. Assume for contradiction that $f(x) = f(x') = v_i$. Then by the construction of a half-edge, we observe that $f(c_i) = f(c'_i) = \beta$. However, this is impossible by the definition of $NAND_2$ gadget.
- for any distinct $i, j \in [k]$ there is $g\colon FF \to H$ such that $g(x) = v_i$, and $g(x') = v_j$. By the construction of a half-edge gadget, there is $f\colon F \to H$, such that $f(x) = v_i$, $f(c_i) = \beta$, and $f(c_{i'}) = \alpha$ for every $i' \neq i$ (in particular, for $i' = j$). Analogously, there is $f'\colon F' \to H$, such that $f'(x') = v_j$, $f'(c_j) = \beta$ and $f'(c_{j'}) = \alpha$ for every $j' \neq j$. We obtain g by combining f and f', and extending this partial homomorphism to vertices of $NAND_2$-gadgets. By the definition of these gadgets, it is possible, as for every i' we have $f(c_{i'}) = \alpha$ or $f'(c_{i'}) = \alpha$.

Observe that the construction so far was performed for H only. Let $g(H)$ be the number of vertices in an edge gadget.

Step 4. Constructing G^* and H-lists L. We start constructing G^* by including the vertex set $V(G)$ of G to G^*(initially they are isolated vertices). For every edge uv of G, we introduce an edge gadget, where x is unified with u and x' is unified with v. By the construction of edge gadget, we observe that for every $v \in V(G)$ we have $L(v) = \mathcal{S}$. Moreover, (G^*, L) is a YES-instance of LHOM(H) if and only if G is k-colorable (we interpret mapping u to $v_i \in \mathcal{S}$ as coloring u with color i). Recall that the size of each edge gadget is $g(H)$, thus the number of vertices of G^* is at most $(|V(G)| + |E(G)|) \cdot g(H)$.

To see that the treewidth of G^* is at most $\mathrm{tw}(G) + g(H)$, consider a tree decomposition $\mathcal{T}$ of G with width $\mathrm{tw}(G)$. For every edge uv of G, we choose one bag X_{uv} of $\mathcal{T}$, such that $u, v \in X_{uv}$. Define a set X'_{uv} as the union of X_{uv} and the set of vertices of the edge gadget corresponding to the edge uv. We extend $\mathcal{T}$ to a tree decomposition $\mathcal{T}^*$ of G^*, by introducing a bag X'_{uv} for every edge uv of G and making it adjacent (in $\mathcal{T}^*$) to X_{uv} only. It is straightforward to verify that $\mathcal{T}^*$ is a tree decomposition of G^* of width at most $\mathrm{tw}(G)+g(H)$. Moreover, it is clear that G^* and L can be constructed in time $g'(H) \cdot poly(|V(G)|)$ for some function g'. Thus, by Theorem 1, our claim holds. ◀

References

1 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74. ACM, 2007. doi:10.1145/1250790.1250801.

2 Hans L. Bodlaender, Paul S. Bonsma, and Daniel Lokshtanov. The fine details of fast dynamic programming over tree decompositions. In Gregory Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*, pages 41–53. Springer, 2013. doi:10.1007/978-3-319-03898-8_5.

3 Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, 2008. doi:10.1093/comjnl/bxm037.

4 Andrei A. Bulatov. H-coloring dichotomy revisited. *Theor. Comput. Sci.*, 349(1):31–39, 2005. doi:10.1016/j.tcs.2005.09.028.

5 Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. Hitting forbidden subgraphs in graphs of bounded treewidth. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, volume 8635 of *Lecture Notes in Computer Science*, pages 189–200. Springer, 2014. doi:10.1007/978-3-662-44465-8_17.

6 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.23.

7 Víctor Dalmau, László Egri, Pavol Hell, Benoit Larose, and Arash Rafiey. Descriptive complexity of list h-coloring problems in logspace: A refined dichotomy. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 487–498. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.52.

8 László Egri, Pavol Hell, Benoit Larose, and Arash Rafiey. Space complexity of list H-colouring: a dichotomy. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth*

Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014, pages 349–365. SIAM, 2014. doi:10.1137/1.9781611973402.26.

9 Tomás Feder and Pavol Hell. List homomorphisms to reflexive graphs. *J. Comb. Theory, Ser. B*, 72(2):236–250, 1998. doi:10.1006/jctb.1997.1812.

10 Tomás Feder, Pavol Hell, and Jing Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19(4):487–505, 1999. doi:10.1007/s004939970003.

11 Tomás Feder, Pavol Hell, and Jing Huang. Bi-arc graphs and the complexity of list homomorphisms. *Journal of Graph Theory*, 42(1):61–80, 2003. doi:10.1002/jgt.10073.

12 Tomás Feder, Pavol Hell, and Jing Huang. List homomorphisms of graphs with bounded degrees. *Discrete Mathematics*, 307(3-5):386–392, 2007. doi:10.1016/j.disc.2005.09.030.

13 Tomás Feder, Pavol Hell, Jing Huang, and Arash Rafiey. Interval graphs, adjusted interval digraphs, and reflexive list homomorphisms. *Discrete Applied Mathematics*, 160(6):697–707, 2012. doi:10.1016/j.dam.2011.04.016.

14 Tomás Feder, Pavol Hell, Sulamita Klein, and Rajeev Motwani. List partitions. *SIAM J. Discrete Math.*, 16(3):449–478, 2003. URL: http://epubs.siam.org/sam-bin/dbq/article/38405.

15 Tomás Feder, Pavol Hell, David G. Schell, and Juraj Stacho. Dichotomy for tree-structured trigraph list homomorphism problems. *Discrete Applied Mathematics*, 159(12):1217–1224, 2011. doi:10.1016/j.dam.2011.04.005.

16 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 142–151. SIAM, 2014. doi:10.1137/1.9781611973402.10.

17 Pavol Hell and Jaroslav Nesetril. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

18 Gábor Kun and Mario Szegedy. A new line of attack on the dichotomy conjecture. *Eur. J. Comb.*, 52:338–367, 2016. doi:10.1016/j.ejc.2015.07.011.

19 C. Lekkeikerker and J. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962. URL: http://eudml.org/doc/213681.

20 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 777–789. SIAM, 2011. doi:10.1137/1.9781611973082.61.

21 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 760–776. SIAM, 2011. doi:10.1137/1.9781611973082.60.

22 Michal Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 520–531. Springer, 2011. doi:10.1007/978-3-642-22993-0_47.

23 Mark H. Siggers. A New Proof of the H-Coloring Dichotomy. *SIAM Journal on Discrete Mathematics*, 23(4):2204–2210, 2010. doi:10.1137/080736697.

24 Zsolt Tuza. Graph colorings with local constraints - a survey. *Discussiones Mathematicae Graph Theory*, 17(2):161–228, 1997. doi:10.7151/dmgt.1049.

25 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.

Small Resolution Proofs for QBF using Dependency Treewidth

Eduard Eiben
Algorithms and Complexity Group, TU Wien, Vienna, Austria and
Department of Informatics, University of Bergen, Norway
eduard.eiben@uib.no

Robert Ganian
Algorithms and Complexity Group, TU Wien, Vienna, Austria and
FI MU, Brno, Czech Republic
rganian@gmail.com

Sebastian Ordyniak
Algorithms and Complexity Group, TU Wien, Vienna, Austria
sordyniak@gmail.com

Abstract

In spite of the close connection between the evaluation of quantified Boolean formulas (QBF) and propositional satisfiability (SAT), tools and techniques which exploit structural properties of SAT instances are known to fail for QBF. This is especially true for the structural parameter treewidth, which has allowed the design of successful algorithms for SAT but cannot be straightforwardly applied to QBF since it does not take into account the interdependencies between quantified variables.

In this work we introduce and develop dependency treewidth, a new structural parameter based on treewidth which allows the efficient solution of QBF instances. Dependency treewidth pushes the frontiers of tractability for QBF by overcoming the limitations of previously introduced variants of treewidth for QBF. We augment our results by developing algorithms for computing the decompositions that are required to use the parameter.

2012 ACM Subject Classification Theory of computation → Logic, Theory of computation → Fixed parameter tractability

Keywords and phrases QBF, Treewidth, Fixed Parameter Tractability, Dependency Schemes

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.28

Related Version The full version of this paper is available at `https://arxiv.org/abs/1711.02120`, [13].

Funding Eduard Eiben was supported by Pareto-Optimal Parameterized Algorithms (ERC Starting Grant 715744) and by the Austrian Science Fund (FWF, projects P26696 and W1255-N23).

1 Introduction

The problem of evaluating quantified Boolean formulas (QBF) is a generalization of the propositional satisfiability problem (SAT) which naturally captures a range of computational tasks in areas such as verification, planning, knowledge representation and automated reasoning [11, 20, 24, 25]. QBF is the archetypical PSPACE-complete problem and is therefore believed to be computationally harder than NP-complete problems such as SAT [18, 21, 31].

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 28; pp. 28:1–28:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In spite of the close connection between QBF and SAT, many of the tools and techniques which work for SAT are known not to help for QBF, and dynamic programming based on the structural parameter treewidth [2, 32] is perhaps the most prominent example of this behavior. Treewidth is a highly-established measure of how "treelike" an instance is, and in the SAT setting it is known that n-variable instances of treewidth at most k can be solved in time at most $f(k) \cdot n$ [32] for a computable function f. Algorithms with running time in this form (i.e., $f(k) \cdot n^{\mathcal{O}(1)}$, where k is the parameter and the degree of the polynomial of n is independent of k) are called *fixed-parameter algorithms*, and problems which admit such an algorithm (w.r.t. a certain parameter) belong to the class *FPT*. Furthermore, in the SAT setting, treewidth allows us to do more than merely solve the instance: it is also possible to find a so-called *resolution proof* [8, 5]. If the input was a non-instance, such a resolution proof contains additional information on "what makes it unsatisfiable" and hence can be more useful than outputting a mere **Reject** in practical settings.

In the QBF setting, the situation is considerably more complicated. It is known that QBF instances of bounded treewidth remain PSpace-complete [2], and the intrinsic reason for this fact is that treewidth does not take into account the dependencies that arise between variables in QBF. So far, there have been several attempts at remedying this situation by introducing variants of treewidth which support fixed-parameter algorithms for QBF: *prefix pathwidth* (along with *prefix treewidth*) [12] and *respectful treewidth* [2], along with two other parameters [1, 6] which originate from a different setting but can also be adapted to obtain fixed-parameter algorithms for QBF. We refer to Subsection 3.2 for a comparison of these parameters. Aside from algorithms with runtime guarantees, it is worth noting that empirical connections between treewidth and QBF have also been studied in the literature [22, 23].

In this work we introduce and develop dependency treewidth, a new structural parameter based on treewidth which supports fixed-parameter algorithms for QBF. Dependency treewidth pushes the frontiers of tractability for QBF by overcoming the limitations of both the previously introduced prefix and respectful variants. Compared to the former, this new parameter allows the computation of resolution proofs analogous to the case of classical treewidth for SAT instances. Prefix pathwidth relies on entirely different techniques to solve QBF and does not yield small resolution proofs. Moreover, the running time of the fixed-parameter algorithm which uses prefix pathwidth has a triple-exponential dependency on the parameter k, while dependency treewidth allows a $\mathcal{O}(3^{2^k} nk)$-time algorithm for QBF.

Unlike respectful treewidth and its variants, which only take the basic dependencies between variables into account, dependency treewidth can be used in conjunction with the so-called *dependency schemes* introduced by Samer and Szeider [26, 29], see also the work of Biere and Lonsing [3]. Dependency schemes allow an in-depth analysis of how the assignment of individual variables in a QBF depends on other variables, and research in this direction has uncovered a large number of distinct dependency schemes with varying complexities. The most basic dependency scheme is called the *trivial dependency scheme* [26], which stipulates that each variable depends on all variables with distinct quantification which precede it in the prefix. Respectful treewidth in fact coincides with dependency treewidth when the trivial dependency scheme is used, but more advanced dependency schemes allow us to efficiently solve instances which otherwise remain out of the reach of state-of-the-art techniques.

Crucially, all of the structural parameters mentioned above require a so-called *decomposition* in order to solve QBF; computing these decompositions is typically an NP-hard problem. A large part of our technical contribution lies in developing algorithms to compute decompositions for dependency treewidth. Without such algorithms, it would not be possible to use the parameter unless a decomposition were supplied as part of the input (an unreal-

istic assumption in practical settings). It is worth noting that all of these algorithms can also be used to find respectful tree decompositions, where the question of finding suitable decompositions was left open [2]. We provide two algorithms for computing dependency tree decompositions, each suitable for use under different situations.

The article is structured as follows. After the preliminaries, we introduce the parameter and show how to use it to solve QBF. This section also contains an in-depth overview and comparison of previous work in the area. A separate section then introduces other equivalent characterizations of dependency treewidth. The last technical section contains our algorithms for finding dependency tree decompositions, after which we provide concluding notes.

2 Preliminaries

For $i \in \mathbb{N}$, we let $[i]$ denote the set $\{1, \ldots, i\}$. We refer to the book by Diestel [9] for standard graph terminology. Given a graph G, we denote by $V(G)$ and $E(G)$ its vertex and edge set, respectively. We use ab as a shorthand for the edge $\{a, b\}$. For $V' \subseteq V(G)$, the *guards* of V' (denoted $\delta(V')$) are the vertices in $V(G) \setminus V'$ with at least one neighbor in V'.

We refer to the standard textbooks [10, 15] for an in-depth overview of parameterized complexity theory. Here, we only recall that a *parameterized problem* (Q, κ) is a *problem* $Q \subseteq \Sigma^*$ together with a *parameterization* $\kappa\colon \Sigma^* \to \mathbb{N}$, where Σ is a finite alphabet. A parameterized problem (Q, κ) is *fixed-parameter tractable (w.r.t. κ)*, in short *FPT*, if there exists a decision algorithm for Q, a computable function f, and a polynomial function p, such that for all $x \in \Sigma^*$, the running time of the algorithm on x is at most $f(\kappa(x)) \cdot p(|x|)$. Algorithms with this running time are called *fixed-parameter algorithms*.

2.1 Quantified Boolean Formulas

For a set of propositional variables K, a *literal* is either a variable $x \in K$ or its negation $\bar{x}$. A *clause* is a disjunction over literals. A *propositional formula in conjunctive normal form* (i.e., a *CNF formula*) is a conjunction over clauses. Given a CNF formula ϕ, we denote the set of variables which occur in ϕ by $\mathrm{var}(\phi)$. For notational purposes, we will view a clause as a set of literals and a CNF formula as a set of clauses.

A *quantified Boolean formula* is a tuple (ϕ, τ) where ϕ is a CNF formula and τ is a sequence of quantified variables, denoted $\mathrm{var}(\tau)$, which satisfies $\mathrm{var}(\tau) \supseteq \mathrm{var}(\phi)$; then ϕ is called the *matrix* and τ is called the *prefix*. A QBF (ϕ, τ) is true if the formula $\tau\phi$ is true. A *quantifier block* is a maximal sequence of consecutive variables with the same quantifier. An *assignment* is a mapping from (a subset of) the variables to $\{0, 1\}$.

The *primal graph* of a QBF $I = (\phi, \tau)$ is the graph G_I defined as follows. The vertex set of G_I consists of every variable which occurs in ϕ, and st is an edge in G_I if there exists a clause in ϕ containing both s and t.

2.2 Dependency Posets for QBF

Before proceeding, we define a few standard notions related to posets which will be used throughout the paper. A partially ordered set (*poset*) $\mathcal{V}$ is a pair $(V, \leq^V)$ where V is a set and $\leq^V$ is a reflexive, antisymmetric, and transitive binary relation over V. A *chain* W of $\mathcal{V}$ is a subset of V such that $x \leq^V y$ or $y \leq^V x$ for every $x, y \in W$. A *chain partition* of $\mathcal{V}$ is a tuple $(W_1, \ldots, W_k)$ such that $\{W_1, \ldots, W_k\}$ is a partition of V and for every i with $1 \leq i \leq k$ the poset induced by W_i is a chain of $\mathcal{V}$. An *anti-chain* A of $\mathcal{V}$ is a subset of V such that for all $x, y \in A$ neither $x \leq^V y$ nor $y \leq^V x$. The *width* (or *poset-width*) of a poset $\mathcal{V}$,

denoted by width($\mathcal{V}$), is the maximum cardinality of any anti-chain of $\mathcal{V}$. A poset of width 1 is called a *linear order*. A *linear extension* of a poset $\mathcal{P} = (P, \leq^P)$ is a relation $\preceq$ over P such that $x \preceq y$ whenever $x \leq^P y$ and the poset $\mathcal{P}^* = (P, \preceq)$ is a linear order. A subset A of V is *downward-closed* if for every $a \in A$ it holds that $b \leq^V a \implies b \in A$. A *reverse* of a poset is obtained by reversing each relation in the poset. For brevity we will often write $\leq^V$ to refer to the poset $\mathcal{V} := (V, \leq^V)$.

We use *dependency posets* to provide a general and formal way of speaking about the various *dependency schemes* introduced for QBF [26]. It is important to note that dependency schemes in general are too broad a notion for our purposes; for instance, it is known that some dependency schemes do not even give rise to sound resolution proof systems. Here we focus solely on so-called *permutation dependency schemes* [28], which is a general class containing all commonly used dependency schemes that give rise to sound resolution proof systems. This leads us to our definition of dependency posets, which allow us to capture all permutation dependency schemes.

Given a QBF $I = (\phi, \tau)$, a *dependency poset* $\mathcal{V} = (\text{var}(\phi), \leq^I)$ of I is a poset over var(ϕ) with the following properties:

1. for all $x, y \in \text{var}(\phi)$, if $x \leq^I y$, then x is before y in the prefix, and
2. given any linear extension $\preceq$ of $\mathcal{V}$, the QBF $I' = (\phi, \tau_\preceq)$, obtained by permutation of the prefix τ according the $\preceq$, is true iff I is true.

The *trivial dependency scheme* is one specific example of a permutation dependency scheme. This gives rise to the *trivial dependency poset*, which sets $x \leq y$ whenever x, y are in different quantifier blocks and x is before y in the prefix. However, more refined permutation dependency schemes which give rise to other dependency posets are known to exist and can be computed efficiently [26, 28]. In particular, it is easy to verify that a dependency poset can be computed from any permutation dependency scheme in polynomial time.

To illustrate these definitions, consider the following QBF: $\exists a \forall b \exists c (a \vee c) \wedge (b \vee c)$. Then the trivial dependency poset would set $a \leq b \leq c$. However, for instance the resolution path dependency poset (arising from the resolution path dependency scheme [33, 27]) contains a single relation $b \leq c$ (in this case, a is incomparable to both b and c).

2.3 Q-resolution

Q-resolution is a sound and complete resolution system for QBF [17]. Our goal here is to formalize the required steps for the Davis Putnam variant of Q-resolution.

We begin with a bit of required notation. For a QBF $I = (\phi, \tau)$ and a variable $x \in \text{var}(\phi)$, let ϕ_x be the set of all clauses in ϕ containing the literal x and similarly let $\phi_{\bar{x}}$ be the set of all clauses containing literal $\bar{x}$. We denote by $res(I, x)$ the QBF $I' = (\phi', \tau')$ such that $\tau' = \tau \setminus \{x\}$ and $\phi' = \phi \setminus (\phi_x \cup \phi_{\bar{x}}) \cup \{(D \setminus \{x\}) \cup (C \setminus \{\bar{x}\}) | D \in \phi_x; C \in \phi_{\bar{x}}\}$; informally, the two clause-sets are pairwise merged to create new clauses which do not contain x. For a QBF $I = (\phi, \tau)$ and a variable $x \in \text{var}(\phi)$ we denote by $I \setminus x$ the QBF $I = (\phi', \tau \setminus \{x\})$, where we get ϕ' from ϕ by removing all occurrences of x and $\bar{x}$.

▶ **Lemma 1.** *Let $I = (\phi, \tau)$ and $x \in var(\phi)$ be the last variable in τ. If x is existentially quantified, then I is true if and only if $res(I, x)$ is true.*

▶ **Lemma 2.** *Let $I = (\phi, \tau)$ and $x \in var(\phi)$ be the last variable in τ. If x is universally quantified, then I is true if and only if $I \setminus x$ is true.*

2.4 Treewidth

Here we will introduce three standard characterizations of treewidth [19]: tree decompositions, elimination orderings, and cops and robber games. These will play a role later on, when we define their counterparts in the dependency treewidth setting and use these in our algorithms.

Tree decomposition. A tree decomposition of a graph G is a pair (T, χ), where T is a rooted tree and χ is a function from $V(T)$ to subsets of $V(G)$, called a *bag*, such that the following properties hold: (T1) $\bigcup_{t \in V(T)} \chi(t) = V(G)$, (T2) for each $uv \in E(G)$ there exists $t \in V(T)$ such that $u, v \in \chi(t)$, and (T3) for every $u \in V(G)$, the set $T_u = \{t \in V(T) : u \in \chi(t)\}$ induces a connected subtree of T.

To distinguish between the vertices of the tree T and the vertices of the graph G, we will refer to the vertices of T as *nodes*. The *width* of the tree decomposition $\mathcal{T}$ is $\max_{t \in T} |\chi(t)| - 1$. The *treewidth* of G, $\mathrm{tw}(G)$, is the minimum width over all tree decompositions of G.

Elimination ordering. An *elimination ordering* of a graph is a linear order of its vertices. Given an elimination ordering ϕ of the graph G, the *fill-in graph* H of G w.r.t. ϕ is the unique minimal graph such that: $V(G) = V(H)$, $E(H) \supseteq E(G)$, and if $0 \leq k < i < j \leq n$ and $v_i, v_j \in N_H(v_k)$, then $v_i v_j \in E(H)$. The *width* of elimination ordering ϕ is the maximum number of neighbors of any vertex v that are larger than v (w.r.t. ϕ) in H.

(Monotone) cops and robber game. The *cops and robber game* is played between two players (the cop-player and the robber-player) on a graph G. A *position* in the game is a pair (C, R) where $C \subseteq V(G)$ is the position of the cop-player and R is a (possibly empty) connected component of $G \setminus C$ representing the position of the robber-player. A move from position (C, R) to position (C', R') is *legal* if it satisfies the following conditions:
CM1 R and R' are contained in the same component of $G \setminus (C \cap C')$,
CM2 $\delta(R) \subseteq C'$.
A play $\mathcal{P}$ is a sequence $(\emptyset, R_0), \ldots, (C_n, R_n)$ of positions such that for every i with $1 \leq i < n$ it holds that the move from (C_i, R_i) to (C_{i+1}, R_{i+1}) is legal; the cop-number of a play is $\max_{i \leq n} |C_i|$. A play $\mathcal{P}$ is won by the cop-player if $R_n = \emptyset$, otherwise it is won by the robber-player. The cop-number of a strategy for the cop player is maximum cop-number over all plays that can arise from this strategy. Finally, the cop-number of G is the minimum cop-number of a winning strategy for the cop player.

For any graph G it holds that G has treewidth k iff G has an elimination ordering of width k iff G has cop-number k [19].

3 Dependency Treewidth for QBF

We are now ready to introduce our parameter. We remark that in the case of dependency treewidth, it is advantageous to start with a counterpart to the elimination ordering characterization of classical treewidth, as this is used extensively in our algorithm for solving QBF. We provide other equivalent characterizations of dependency treewidth (representing the counterparts to tree decompositions and cops and robber games) in Section 4; these are not only theoretically interesting, but serve an important role in our algorithms for computing the dependency treewidth. Furthermore, we remark that on existentially quantified QBFs dependency treewidth w.r.t. trivial dependency poset collapses with classical treewidth.

Let $I = (\phi, \tau)$ be a QBF instance with a dependency poset $\mathcal{P}$. An elimination ordering of G_I is *compatible* with $\mathcal{P}$ if it is a linear extension of the reverse of $\mathcal{P}$; intuitively, this

corresponds to being forced to eliminate variables that have the most dependencies first. For instance, if $\mathcal{P}$ is a trivial dependency poset then a compatible elimination ordering must begin eliminating from the rightmost block of the prefix. We call an elimination ordering of G_I that is compatible with $\mathcal{P}$ a *$\mathcal{P}$-elimination ordering* (or *dependency elimination ordering*). The *dependency treewidth* w.r.t. $\mathcal{P}$ is then the minimum width of a $\mathcal{P}$-elimination ordering.

3.1 Using dependency treewidth

Our first task is to show how dependency elimination orderings can be used to solve QBF.

▶ **Theorem 3.** *There is an algorithm that given* 1. *a QBF I with n variables and m clauses,* 2. *a dependency poset $\mathcal{P}$ for I, and* 3. *a $\mathcal{P}$-elimination ordering π of width k, decides whether I is true in time $\mathcal{O}(3^{2k}kn)$. Moreover, if I is false, then the algorithm outputs a Q-resolution refutation of size $\mathcal{O}(3^k n)$.*

Sketch of Proof. Let $I = (\phi, \tau)$ and let $x_1, \ldots, x_n$ denote the variables of ϕ such that $x_i \leq_\pi x_{i+1}$ for all $1 \leq i < n$. From the definition of the dependency poset and the fact that π is a dependency elimination ordering, it follows that the QBF instance $I' = (\phi, \tau')$, where τ' is the reverse of π, is true if and only if I is true.

To solve I we use a modification of the Davis Putnam resolution algorithm [8]. We start with instance I' and recursively eliminate the last variable in the prefix using Lemmas 1 and 2 until we either run out of variables or we introduce as a resolvent a non-tautological clause that is either empty or contains only universally quantified variables. We show that each variable we eliminate has the property that it only shares clauses with at most k other variables, and in this case we introduce at most 3^k clauses of size at most k at each step.

From now on let H be the fill-in graph of the primal graph of I with respect to π, and let us define $I_i = (\phi^i, \tau^i)$ for $1 \leq i \leq n$ as follows: (1) $I_1 = I'$, (2) $I_{i+1} = I_i \setminus x_i$ if x_i is universally quantified, and (3) $I_{i+1} = res(I_i, x_i)$, if x_i is existentially quantified.

Note that x_i is always the last variable of the prefix of I_i and it follows from Lemmas 1 and 2 that I_{i+1} is true if and only if I_i is true. Moreover, I_n only contains a single variable, and hence can be decided in constant time. One can show by induction that I_{i+1} contains at most 3^k new clauses, i.e., clauses not contained in I_i. To this end, we show and use the fact that both ϕ^i_x and $\phi^i_{\bar{x}}$ contain at most 3^k clauses, and this is sufficient to ensure a small Q-resolution refutation if the instance is false. A formal proof of this fact and runtime analysis are provided in the full version. ◀

3.2 A Comparison of Decompositional Parameters for QBF

As was mentioned in the introduction, two dedicated decompositional parameters have previously been introduced specifically for evaluating quantified Boolean formulas: *prefix pathwidth* (and, more generally, *prefix treewidth*) [12] and *respectful treewidth* [2]. The first task of this section is to outline the advantages of dependency treewidth compared to these two parameters. We remark that we do not include formal definitions of the parameters in this section, since they are technical and not critical for our exposition.

Prefix pathwidth is based on bounding the number of viable strategies in the classical two-player game characterization of the QBF problem [12]. As such, it decomposes the dependency structure of a QBF instance *beginning from variables that have the least dependencies* (i.e., may appear earlier in the prefix). On the other hand, our dependency treewidth is based on Q-resolution and thus decomposes the dependency structure *beginning from variables that have the most dependencies* (i.e., may appear last in the prefix). Lemma 4 shows that both

approaches are, in principle, incomparable. That being said, dependency treewidth has two critical advantages over prefix treewidth/pathwidth:

1. dependency treewidth outputs small resolution proofs, while it is not at all clear whether the latter can be used to obtain such resolution proofs;
2. dependency treewidth supports a single-exponential fixed-parameter algorithm for QBF (Theorem 3), while the latter uses a prohibitive triple-exponential algorithm [12].

▶ **Lemma 4.** *Let us fix the trivial dependency poset. There exist infinite classes $\mathcal{A}, \mathcal{B}$ of QBF instances such that:*

- **a.** *$\mathcal{A}$ has unbounded dependency treewidth but prefix pathwidth at most 1;*
- **b.** *$\mathcal{B}$ has unbounded prefix pathwidth (and prefix treewidth) but dependency treewidth at most 1.*

Sketch of Proof. Consider the following examples for $\mathcal{A}, \mathcal{B}$. Let $\mathcal{A} = \{A_i = \exists x_1, \ldots, x_i \forall y \exists x (y \vee x) \wedge \bigwedge_{j=1}^{i}(x_j \vee x)\}$, and let $\mathcal{B} = \{B_i = \exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \exists x_{2^i-1} \forall x_{2^i} \exists x_{2^i+1} \bigwedge_{j=1}^{i-1}((x_j \vee x_{2j}) \wedge (x_j \vee x_{2j+1}))\}$. It is straightforward to verify that these classes satisfy the conditions stipulated in the lemma. ◀

Respectful treewidth coincides with dependency treewidth when the trivial dependency scheme is used, i.e., represents a special case of our measure. Unsurprisingly, the use of more advanced dependency schemes (such as the resolution path dependency scheme [33, 28]) allows the successful deployment of dependency treewidth on much more general classes of QBF instances. Furthermore, dependency treewidth with such dependency schemes will always be upper-bounded by respectful treewidth, and so algorithms based on dependency treewidth will outperform the previously introduced respectful treewidth based algorithms.

▶ **Lemma 5.** *There exists an infinite class $\mathcal{C}$ of QBF instances such that $\mathcal{C}$ has unbounded dependency treewidth with respect to the trivial dependency poset but dependency treewidth at most 1 with respect to the resolution-path dependency poset.*

Sketch of Proof. It suffices to set $\mathcal{C}$ to be equal to the class $\mathcal{A}$ used in the proof of the previous lemma and then verify that $\mathcal{C}$ has the desired properties. ◀

Finally, we note that the idea of exploiting dependencies among variables has also given rise to similarly flavored structural measures in the areas of first-order model checking (first order treewidth) [1] and quantified constraint satisfaction (CD-width[1]) [6]. Even though the settings differ, Theorem 5.5 [1] and Theorem 5.1 [6] can both be translated to a basic variant of Theorem 3. We note that this readily-obtained variant of Theorem 3 would not account for dependency schemes. We conclude this subsection with two lemmas which show that there are classes of QBF instances that can be handled by our approach but are not covered by the results of Adler, Weyer [1] and Chen, Dalmau [6].

▶ **Lemma 6.** *There exist infinite classes $\mathcal{D}$, $\mathcal{E}$ of QBF instances such that:*

- **a.** *$\mathcal{D}$ has unbounded CD-width but dependency treewidth at most 1 w.r.t. the resolution-path dependency poset.*
- **b.** *$\mathcal{E}$ has unbounded dependency treewidth w.r.t. any dependency poset but CD-width at most 1.*

[1] We remark that in their paper, the authors refer to their parameter simply as "the width". For disambiguation, here we call it CD-width (shorthand for Chen-Dalmau's width).

Sketch of Proof. It suffices to set $\mathcal{D}$ to be equal to the class $\mathcal{A}$ used in the proof of Lemma 4 and then observe that the same class of instances is used as an example of a class with unbounded CD-width by Chen, Dalmau [6, Example 3.6]. On the other hand, it easy to verify that the QBF instance $E_i = \forall x_1 \forall x_2 \cdots \forall x_i \bigwedge_{1 \leq p < q \leq i} (x_p \vee x_q)$ has CD-width 0, as E_i does not contain an existential variable. However, the primal graph of E_i is a clique and hence it has dependency treewidth $i - 1$. ◀

▶ **Lemma 7.** *There exists an infinite class $\mathcal{F}$ of QBF instances such that $\mathcal{F}$ has unbounded first order treewidth but dependency treewidth at most 2 with respect to the resolution-path dependency poset.*

Sketch of Proof. Let $F_i = \forall x_{2i} y_{2i} \exists x_{2i-1} y_{2i-1} \cdots \forall x_2 y_2 \exists x_1 y_1 \forall z (z \vee x_1 \vee y_1) \bigwedge_{j=1}^{2i-1} [(x_j \vee x_{j+1}) \wedge (y_j \vee y_{j+1})] \wedge \bigwedge_{j=1}^{i} (x_{2j-1} \vee y_1)$. It is readily observed that the elimination ordering $z x_1 x_2 \ldots x_{2i} y_1 y_2 \ldots y_{2i}$ of width 2 is compatible with the resolution-path dependency poset for this formula. On the other hand, the elimination ordering obtained from first order treewidth is forced to eliminate y_1 before x_j, $j \geq 2$ (see Definitions 3.3, 3.10, 3.15, together with the definitions on page 5 of Adler and Weyer [1]). Therefore, the first order treewidth of this instance would be at least $i - 1$. ◀

4 Dependency Treewidth: Characterizations

In this section we obtain other equivalent characterizations of dependency treewidth. The purpose of this endeavor is twofold. From a theoretical standpoint, having several natural characterizations (corresponding to the characterizations of treewidth) is not only interesting but also, in some sense, highlights the solid foundations of a structural parameter. From a practical standpoint, the presented characterizations play an important role in Section 5, which is devoted to algorithms for finding optimal dependency elimination orderings.

Dependency tree decomposition. Let I be a QBF instance with primal graph G and dependency poset $\mathcal{P}$ and let (T, χ) be a tree decomposition of G. Note that the rooted tree T naturally induces a partial order $\leq_T$ on its nodes, where the smallest element is the root and leaves form maximal elements. For a vertex $v \in V(G)$, we denote by $F_v(T)$ the unique $\leq_T$-minimal node t of T with $v \in \chi(t)$, which is well-defined because of Properties (T1) and (T3) of a tree decomposition. Let $<_{\mathcal{T}}$ be the partial ordering of $V(G)$ such that $u <_{\mathcal{T}} v$ if and only if $F_u(T) <_T F_v(T)$ for every $u, v \in V(G)$. We say that (T, χ) is a *dependency tree decomposition* if it satisfies the following additional property:

(T4) $<_{\mathcal{T}}$ is compatible with $\leq^{\mathcal{P}}$, i.e., for every two vertices u and v of G it holds that whenever $F_u(T) <_T F_v(T)$ then it does not hold that $v \leq^{\mathcal{P}} u$.

▶ **Lemma 8.** *A graph G has a $\mathcal{P}$-elimination ordering of width at most ω if and only if G has a dependency tree decomposition of width at most ω. Moreover, a $\mathcal{P}$-elimination ordering of width ω can be obtained from a dependency tree decomposition of width ω in polynomial-time and vice versa.*

Proof. For the forward direction we will employ the construction given by Kloks in [19], which shows that a normal elimination ordering can be transformed into a tree decomposition of the same width. We will then show that this construction also retains the compatibility with $\mathcal{P}$. Let $\leq^{\phi} = (v_1, \ldots, v_n)$ be a dependency elimination ordering for G of width ω and let H be the fill-in graph of G w.r.t. $\leq^{\phi}$. We will iteratively construct a sequence $(\mathcal{T}_0, \ldots, \mathcal{T}_{n-1})$ such that for every i with $0 \leq i < n$, $\mathcal{T}_i = (T_i, \chi_i)$ is dependency tree decompositions of

the graph $H_i = H[\{v_{n-i}, \ldots, v_n\}]$ of width at most ω. Because $\mathcal{T}_{n-1}$ is a dependency tree decomposition of $H_{n-1} = H$ of width at most ω, this shows the forward direction of the lemma. In the beginning we set $\mathcal{T}_0$ to be the trivial tree decomposition of H_0, which contains merely one node whose bag consists of the vertex v_n. Moreover, for every i with $0 < i < n$, $\mathcal{T}_i$ is obtained from $\mathcal{T}_{i-1}$ as follows. Note that because $N_{H_i}(v_{n-i})$ induces a clique in H_{i-1}, $\mathcal{T}_{i-1}$ contains a node that covers all vertices in $N_{H_i}(v_{n-i})$. Let t be any such bag, then is $\mathcal{T}_i$ is obtained from $\mathcal{T}_{i-1}$ by adding a new node t' to T_{i-1} making it adjacent to t and setting $\chi_i(t') = N_{H_i}[v_{n-i}]$. It is known [19] that $\mathcal{T}_i$ satisfies the Properties (T1)–(T3) of a tree decomposition and it hence only remains to show that $\mathcal{T}_i$ satisfies (T4). Since, by induction hypothesis, $\mathcal{T}_{i-1}$ is a dependency tree decomposition, Property (T4) already holds for every pair $u, v \in V(H_{i-1})$. Hence it only remains to consider pairs u and v_{n-i} for some $u \in V(H_{i-1})$. Because the only node containing v_{n-i} in $\mathcal{T}_i$ is a leaf, we can assume that $F_u(T) <_T F_{v_{n-i}}(T)$ and because $v_{n-i} \leq^{\phi} u$ it cannot hold that $v_{n-i} \leq^{\mathcal{P}} u$, as required.

For the reverse direction, let $\mathcal{T} = (T, \chi)$ be a $\mathcal{P}$-tree decomposition of G of width at most ω. It is known [19] that any linear extension of $<_{\mathcal{T}}$ is an elimination ordering for G of width at most ω. Moreover, because of Property (T4), $<_{\mathcal{T}}$ is compatible with $\leq^{\mathcal{P}}$ and hence there is a linear extension of $<_{\mathcal{T}}$, which is also a linear extension of the reverse of $\leq^{\mathcal{P}}$. ◀

Dependency cops and robber game. Recalling the definition of the (monotone) cops and robber game for treewidth, we define the *dependency cops and robber game* (for a QBF instance I with dependency poset $\mathcal{P}$) analogously but with the additional restriction that legal moves must also satisfy a third condition:

CM3 $C' \setminus C$ is downward-closed in R, i.e., there is no $r \in R \setminus C'$ with $r \leq^{\mathcal{P}} c$ for any $c \in C' \setminus C$.

Intuitively, condition CM3 restricts the cop-player by forcing him to search vertices (variables) in an order that is compatible with the dependency poset.

To formally prove the equivalence between the cop-number for this restricted game and dependency treewidth, we will need to also formalize the notion of a strategy. Here we will represent strategies for the cop-player as rooted trees whose nodes are labeled with positions for the cop-player and whose edges are labeled with positions for the robber-player. Namely, we will represent winning strategies for the cop-player on a primal graph G by a triple (T, α, β), where T is a rooted tree, $\alpha : V(T) \to 2^{V(G)}$ is a mapping from the nodes of T to subsets of $V(G)$, and $\beta : E(T) \to 2^{V(G)}$, satisfying the following conditions:

CS1 $\alpha(r) = \emptyset$ and for every component R of G, the root node r of T has a unique child c with $\beta(\{r, c\}) = R$, and

CS2 for every other node t of T with parent p it holds that: the move from position $(\alpha(p), \beta(\{p, t\}))$ to position $(\alpha(t), \beta(\{t, c\}))$ is legal for every child c of t and moreover for every component R of $G \setminus \alpha(t)$ contained in $\beta(\{p, t\})$, t has a unique child c with $\beta(\{t, c\}) = R$.

Informally, the above properties ensure that every play consistent with the strategy is winning for the cop-player and moreover for every counter-move of the robber-player, the strategy gives a move for the cop-player. The width of a winning strategy for the cop-player is the maximum number of cops simultaneously placed on G by the cop-player, i.e., $\max_{t \in V(T)} |\alpha(t)|$. The cop-number of G is the minimum width of a winning strategy for the cop-player on G.

▶ **Lemma 9.** *For every graph G the width of an optimal dependency tree decomposition plus one is equal to the cop-number of the graph. Moreover, a dependency tree decomposition of width ω can be obtained from a winning strategy for the cop-player of width $\omega + 1$ in polynomial-time and vice versa.*

Proof. Let $\mathcal{T} = (T, \chi)$ be a dependency tree decomposition of G of width ω. First, we show that $\mathcal{T}$ can be transformed into a dependency tree decomposition of width ω satisfying:

(*) $\chi(r) = \emptyset$ for the root node r of T and for every node $t \in V(T)$ with child $c \in V(T)$ in T the set $\chi(T_c) \setminus \chi(t)$ is a component of $G \setminus \chi(t)$.

To ensure that $\mathcal{T}$ satisfies $\chi(r) = \emptyset$ it is sufficient to add a new root vertex r' to T and set $\chi(r') = \emptyset$. We show next that starting from the root of T we can ensure that for every node $t \in V(T)$ with child c the set $\chi(T_c) \setminus \chi(t)$ is a component of $G \setminus \chi(t)$. Let t be a node with child c in T for which this does not hold. By the well-known separation property of tree-decompositions, we have that $\chi(T_c) \setminus \chi(t)$ is a set of components, say containing $C_1, \ldots, C_l$, of $G \setminus \chi(t)$. For every i with $1 \leq i \leq l$, let $\mathcal{T}_i = (T_i, \chi_i)$ be the dependency tree decomposition with $T_i = T_c$ and $\chi_i(t') = \chi(t') \cap (C_i \cup \chi(t))$ and root $r_i = c$. Then we replace the entire sub dependency tree decomposition of $\mathcal{T}$ induced by T_c in T with the tree decompositions $\mathcal{T}_1, \ldots, \mathcal{T}_l$ such that t now becomes adjacent to the roots $r_1, \ldots, r_l$. It is straightforward to show that the result of this operation is again a dependency tree decomposition of G of width at most ω and moreover the node t has one child less that violates (*). By iteratively applying this operation to every node t of $\mathcal{T}$ we eventually obtain a dependency tree decomposition that satisfies (*).

Hence w.l.o.g. we can assume that $\mathcal{T}$ satisfies (*). We now claim that (T, α, β) where:

- $\alpha(t) = \chi(t)$ for every $t \in V(T)$,
- for a node $t \in V(T)$ with parent $p \in V(T)$, $\beta(\{p, t\}) = \chi(T_t) \setminus \chi(p)$.

is a winning strategy for $\omega + 1$ cops. Observe that because $\mathcal{T}$ satisfies (*), it holds that $\alpha(r) = \emptyset$ and for every $t \in V(T)$ with parent $p \in V(T)$, the pair $(\alpha(p), \beta(\{p, t\})$ is a position in the visible $\mathcal{P}$-cops and robber game on G. Furthermore, it is possible to verify that for every t, p as above and every child c of t in T, it holds that the move from $(\alpha(p), \beta(\{p, t\})$ to $(\alpha(t), \beta(\{t, c\})$ is valid.

On the other hand, let $\mathcal{S} = (T, \alpha, \beta)$ be a winning strategy for the cop-player in the visible $\mathcal{P}$-cops and robber game on G using ω cops. Observe that $\mathcal{S}$ can be transformed into a winning strategy for the cop-player using ω cops satisfying:

(^{a}) for every node t of T with parent p it holds that $\alpha(t) \subseteq \delta(\beta(\{p, t\})) \cup \beta(\{p, t\})$.

Indeed; if (^{a}) is violated, then one can simply change $\alpha(t)$ to $\alpha(t) \cap (\delta(\beta(\{p, t\})) \cup \beta(\{p, t\}))$ without violating any of CS1 or CS2. Hence we can assume that $\mathcal{S}$ satisfies (^{a}).

We now claim that $\mathcal{T} = (T, \alpha)$ is a dependency tree decomposition of G of width $\omega - 1$. Towards showing T1, let $v \in V(G)$. Because of CS1, it holds that either $v \in \alpha(r)$ for the root r of T or there is a child c of r in T with $v \in \beta(\{r, c\})$. Moreover, due to CS2 we have that either $v \in \alpha(c)$ or $v \in \beta(\{c, c'\})$ for some child c' of c in T. By proceeding along T, we will eventually find a node $t \in V(T)$ with $v \in \alpha(t)$. Towards showing T2, let $\{u, v\} \in E(G)$. Again because of CS1, it holds that either $\{u, v\} \subseteq \alpha(r)$, or $\{u, v\} \subseteq \delta(\beta(\{r, c\})) \cup \beta(\{r, c\})$ for some child c of r in T. Because of CM2, we obtain that $\delta(\beta(\{r, c\})) \subseteq \alpha(c)$ and together with CS2, we have that either $\{u, v\} \subseteq \alpha(c)$ or $\{u, v\} \subseteq \delta(\beta(\{c, c'\})) \cup \beta(\{c, c'\})$ for some child c' of c in T. By proceeding along T, we will eventually find a node $t \in V(T)$ with $\{u, v\} \in \alpha(t)$. Finally, in order to argue that T3 and T4 hold, we will first establish that $\mathcal{S}$ satisfies the following property:

(^{b}) for every node t with child c in T it holds that $\bigcup_{t' \in V(T_c)} \alpha(t') \subseteq \delta(\beta(\{t, c\}) \cup \beta(\{t, c\})$.

Because of CM2 we have that $\beta(\{t, c\}) \subseteq \beta(\{p, t\})$ for every three nodes p, t, and c such that p is the parent of t which in turn is the parent of c in T. Moreover, because of (^{a}) we have that $\alpha(t) \subseteq \delta(\beta(\{p, t\})) \cup \beta(\{p, t\})$ for every node t with parent p in T. Applying these two facts iteratively along a path from t to any of its descendants t' in T, we obtain that $\alpha(t') \subseteq \delta(\beta(\{p, t\})) \cup \beta(\{p, t\})$, as required.

Knowing (^{b}) and (^{a}), it is not too difficult to show that T3 and T4 hold. This means that $\mathcal{T}$ is a dependency tree-decomposition, completing the proof. ◀

5 Computing Dependency Treewidth

In this section we will present two exact algorithms to compute dependency treewidth. The first algorithm is based on the characterization of dependency treewidth in terms of the cops and robber game and shows that, for every fixed ω, determining whether a graph has dependency treewidth at most ω, and in the positive case also computing a dependency tree decomposition of width at most ω, can be achieved in polynomial time. The second algorithm is based on a chain partition of the given dependency poset and shows that if the width of the poset is constant, then an optimal dependency tree decomposition can be constructed in polynomial time.

Before proceeding to the algorithms, we would like to mention here that the fixed-parameter algorithm for computing first order treewidth [1] can also be used for computing dependency treewidth in the restricted case that the trivial dependency poset is used.

▶ **Theorem 10.** *There is an algorithm running in time $\mathcal{O}(|V(G)|^{2\omega+2})$ that, given a graph G and a poset $\mathcal{P} = (V(G), \leq^{\mathcal{P}})$ and $\omega \in \mathbb{N}$, determines whether ω cops have a winning strategy in the dependency cops and robber game on G and $\mathcal{P}$, and if so outputs such a winning strategy.*

Sketch of Proof. This algorithm is similar to the folklore $n^{\mathcal{O}(\omega)}$ algorithm for computing treewidth based on cops and robber game; see, e.g., Exercise 7.26 in Cygan et al. [7]. The idea is to transform the cops and robber game on G into a much simpler two player game, which is played on all possible positions of the cops and robber game on G.

A *simple two player game* is played between two players, which in association to the cops and robber game, we will just call the cops and the robber player [16]. Both players play by moving a token around on a so-called *arena*, which is a triple $\mathcal{A} = (V_C, V_R, A)$ such that $((V_C \cup V_R), A)$ is a bipartite directed graph with bipartition (V_C, V_R). The vertices in V_C are said to belong to the cop-player and the vertices in V_R are said to belong to the robber-player. Initially, one token is placed on a distinguished starting vertex $s \in V_C \cup V_R$. From then onward the player who owns the vertex, say v, that currently contains the token, has to move the token to an arbitrary successor (i.e., out-neighbor) of v in $\mathcal{A}$. The cop-player wins if the robber-player gets stuck, i.e., the token ends up in a vertex owned by the robber-player that has no successors in $\mathcal{A}$, otherwise the robber-player wins. It is well-known that strategies in this game are deterministic and memoryless, i.e., strategies for a player are simple functions that assign every node owned by the player one of its successors. Moreover, the winning region for both players as well as their corresponding winning strategy can be computed in time $\mathcal{O}(|V_C \cup V_R| + |A|)$ by the following algorithm. The algorithm first computes the winning region W_C, as follows.

Initially all vertices owned by the robber-player which do not have any successors in $\mathcal{A}$ are placed in W_C. The algorithm then iteratively adds the following vertices to W_C:

- all vertices owned by the cop-player that have at least one successor W_C,
- all vertices owned by the robber-player for which all successors are in W_C.

Once the above process stops, the set W_C is the winning region of the cop-player in $\mathcal{A}$ and $(V_C \cup V_R) \setminus W_C$ is the winning region for the robber-player. Moreover, the winning strategy for both players can now be obtained by choosing for every vertex a successors that is in the winning region of the player owning that vertex (if no such vertex exists, then an arbitrary successor must be chosen).

Given a graph G, a poset $\mathcal{P} = (V(G), \leq^{\mathcal{P}})$, and an integer ω, we construct an arena $\mathcal{A} = (V_C, V_R, A)$ and a starting vertex $s \in V_R$ such that ω cops have a winning strategy in the $\mathcal{P}$-cops and robber game on G iff the cop-player wins from s in the simple two player game on $\mathcal{A}$ as follows:

- We set V_C to be the set of all pairs (C, R) such that (C, R) is a position in the $\mathcal{P}$-cops and robber game on G using at most ω cops ($|C| \leq \omega$),
- We set V_R to be the set of all triples (C, C', R) such that:
  - (C, R) is a position in the $\mathcal{P}$-cops and robber game on G using at most ω cops, and
  - $C' \subseteq V(G)$ is a potential new cop-position for at most ω cops from (C', R'), i.e., $\delta(R) \subseteq C'$ and C, R, and C' satisfy CM3.
- From every vertex $(C, R) \in V_C$ we add an arc to all vertices $(C, C', R) \in V_R$.
- From every vertex $(C, C', R) \in V_R$ we add an arc to all vertices $(C', R') \in V_C$ such that the move from (C, R) to (C', R') is legal.
- Additionally V_R contains the starting vertex s that has an outgoing arc to every vertex $(\emptyset, R) \in V_C$ such that R is a component of G.

Finally, it is straightforward to show that the cop-player has a winning strategy from s in $\mathcal{A}$ iff G and $\mathcal{P}$ have cop-number at most ω. ◀

The next theorem summarizes our second algorithm for computing dependency treewidth. The core distinction here lies in the fact that the running time does not depend on the dependency treewidth, but rather on the poset-width. This means that the algorithm can precisely compute the dependency treewidth even when this is large, and it will perform better than Theorem 10 for formulas with "tighter" dependency structures (e.g., formulas which utilize the full power of quantifier alternations).

▶ **Theorem 11.** *There is an algorithm running in time $\mathcal{O}((|V(G)|^k k^2)$ that, given a graph G and a poset $\mathcal{P} = (V(G), \leq^{\mathcal{P}})$ of width k and $\omega \in \mathbb{N}$, determines whether G and $\mathcal{P}$ admit a dependency elimination ordering of width at most ω, and if yes outputs such a dependency elimination ordering.*

Proof. To decide whether G has a dependency elimination ordering of width at most ω, we first build an auxiliary directed graph H as follows.

The vertex set of H consists of all pairs (D, d) such $D \subseteq V(G)$ is a downward closed set and $d \in D$ is a maximal element of D such that $|(N_G(C) \cap (D \setminus \{d\})| \leq \omega$, where C is the unique component of $G \setminus (D \setminus \{d\})$ containing d. Note that $(N_G(C) \cap (D \setminus \{d\})$ is equal to the set of neighbors of d in any fill-in graph w.r.t. to any linear order ϕ for which d is larger than all vertices in $V(G) \setminus D$ and smaller than the vertices in $D \setminus \{d\}$. Intuitively, a vertex (D, d) in H corresponds to the step in which we eliminate vertex d after exactly the vertices in $V(G) \setminus D$ have already been eliminated. Additionally, H contains the vertices $(V(G), \emptyset)$ and $(\emptyset, \emptyset)$. Furthermore, there is an arc from (D, d) to (D', d') of H if and only if $D' = D \cup \{d'\}$ or $D = D' = V(G)$ and $d' = \emptyset$. This completes the construction of H. It is immediate that G has a dependency elimination ordering of width at most ω if and only if there is a directed path in H from $(\emptyset, \emptyset)$ to $(V(G), \emptyset)$. Hence, given H we can decide whether G has a dependency elimination ordering of width at most ω (and output it, if it exists) in time $\mathcal{O}(|V(H)| \log(|V(H)|) + E(H))$ (e.g., by using Dijkstra's algorithm). It remains to analyze the time required to construct H (as well as its size).

Let k be the width of the poset $\mathcal{P}$. By the algorithm of Felsner, Raghavan and Spinrad [14], we can compute a chain partition $\mathcal{C} = (W_1, \ldots, W_k)$ of width k in time $\mathcal{O}(k \cdot |V(G)|^2)$. Note that every downward closed D set can be characterized by the position of the maximal

element in D on each of the chains $W_1, \ldots, W_k$, we obtain that there are at most $|V(G)|^k$ downward closed sets. Hence, H has at most $\mathcal{O}(|V(G)|^k(k+1))$ vertices its vertex set can be constructed in time $\mathcal{O}(|V(G)|^k(k+1))$. Since every vertex (D, d) of H has at most $k+1$ possible out-neighbors, we can construct the arc set of H in time $\mathcal{O}(|V(G)|^k k^2)$.

Hence, the total time required to construct H is $\mathcal{O}((|V(G)|^k k^2)$ which dominates the time required to find a shortest path in H, and so the runtime follows. ◀

6 Concluding Notes

Dependency treewidth is a promising decompositional parameter for QBF which overcomes the key shortcomings of previously introduced structural parameters; its advantages include a single-exponential running time, a refined and flexible approach to variable dependencies, and the ability to compute decompositions. It also admits several natural characterizations that show the robustness of the parameter and allows the computation of resolution proofs.

The presented algorithms for computing dependency elimination orderings leave open the question of whether this problem admits a fixed-parameter algorithm (parameterized by dependency treewidth). We note that the two standard approaches for computing treewidth fail here. In particular, the well-quasi-ordering approach with respect to minors does not work since the set of ordered graphs can be observed not to be well-quasi ordered w.r.t. the ordered minor relation [30]. On the other hand, the records used in the second approach [4] do not provide sufficient information in our ordered setting.

References

1. Isolde Adler and Mark Weyer. Tree-width for first order formulae. *Logical Methods in Computer Science*, 8(1), 2012.
2. Albert Atserias and Sergi Oliva. Bounded-width QBF is pspace-complete. *J. Comput. Syst. Sci.*, 80(7):1415–1429, 2014.
3. Armin Biere and Florian Lonsing. Integrating dependency schemes in search-based QBF solvers. In *Theory and Applications of Satisfiability Testing - SAT 2010*, volume 6175 of *Lecture Notes in Computer Science*, pages 158–171. Springer, 2010.
4. Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 226–234, 1993.
5. Florent Capelli. *Structural restrictions of CNF-formulas: applications to model counting and knowledge compilation.* PhD thesis, Université Paris Diderot, 2016.
6. Hubie Chen and Víctor Dalmau. Decomposing quantified conjunctive (or disjunctive) formulas. *SIAM J. Comput.*, 45(6):2066–2086, 2016.
7. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
8. M. Davis and H. Putnam. A computing procedure for quantification theory. 7(3):201–215, 1960.
9. Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
10. Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer Verlag, 2013.
11. Uwe Egly, Thomas Eiter, Hans Tompits, and Stefan Woltran. Solving advanced reasoning tasks using quantified boolean formulas. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.*, pages 417–422, 2000.

12 Eduard Eiben, Robert Ganian, and Sebastian Ordyniak. Using decomposition-parameters for QBF: mind the prefix! In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 964–970, 2016.

13 Eduard Eiben, Robert Ganian, and Sebastian Ordyniak. Small resolution proofs for QBF using dependency treewidth. *CoRR*, abs/1711.02120, 2017. `arXiv:1711.02120`.

14 S. Felsner, V. Raghavan, and J. Spinrad. Recognition algorithms for orders of small width and graphs of small dilworth number. *Order*, 20(4):351–364, 2003.

15 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer Verlag, Berlin, 2006.

16 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

17 Hans Kleine Büning and Uwe Bubeck. Theory of quantified boolean formulas. In A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 23, pages 735–760. IOS Press, 2009.

18 Hans Kleine Büning and Theodor Lettman. *Propositional logic: deduction and algorithms*. Cambridge University Press, Cambridge, 1999.

19 T. Kloks. *Treewidth: Computations and Approximations*. Springer Verlag, Berlin, 1994.

20 Charles Otwell, Anja Remshagen, and Klaus Truemper. An effective QBF solver for planning problems. In *Proceedings of the International Conference on Modeling, Simulation & Visualization Methods, MSV '04 & Proceedings of the International Conference on Algorithmic Mathematics & Computer Science, AMCS '04, June 21-24, 2004, Las Vegas, Nevada, USA*, pages 311–316. CSREA Press, 2004.

21 Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

22 Luca Pulina and Armando Tacchella. A structural approach to reasoning with quantified boolean formulas. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 596–602, 2009.

23 Luca Pulina and Armando Tacchella. An empirical study of QBF encodings: from treewidth estimation to useful preprocessing. *Fundam. Inform.*, 102(3-4):391–427, 2010.

24 J. Rintanen. Constructing conditional plans by a theorem-prover. *J. Artif. Intell. Res.*, 10:323–352, 1999.

25 Ashish Sabharwal, Carlos Ansótegui, Carla P. Gomes, Justin W. Hart, and Bart Selman. QBF modeling: Exploiting player symmetry for simplicity and efficiency. In *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, pages 382–395, 2006.

26 Marko Samer and Stefan Szeider. Backdoor sets of quantified Boolean formulas. *Journal of Autom. Reasoning*, 42(1):77–97, 2009.

27 Friedrich Slivovsky and Stefan Szeider. Computing resolution-path dependencies in linear time. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012*, volume 7317 of *Lecture Notes in Computer Science*, pages 58–71. Springer Verlag, 2012.

28 Friedrich Slivovsky and Stefan Szeider. Quantifier reordering for QBF. *J. Autom. Reasoning*, 56(4):459–477, 2016.

29 Friedrich Slivovsky and Stefan Szeider. Soundness of q-resolution with dependency schemes. *Theor. Comput. Sci.*, 612:83–101, 2016.

30 Daniel A Spielman and Miklós Bóna. An infinite antichain of permutations. *Electron. J. Combin*, 7:N2, 2000.

31 L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 1–9. ACM, 1973.

32 Stefan Szeider. On fixed-parameter tractable parameterizations of SAT. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability, 6th International Conference, SAT 2003, Selected and Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 188–202. Springer Verlag, 2004.

33 Allen Van Gelder. Variable independence and resolution paths for quantified boolean formulas. In Jimmy Lee, editor, *Principles and Practice of Constraint Programming - CP 2011*, volume 6876 of *Lecture Notes in Computer Science*, pages 789–803. Springer Verlag, 2011.

Lossy Kernels for Connected Dominating Set on Sparse Graphs

Eduard Eiben
Algorithms and Complexity Group, TU Wien, Austria and
Department of Informatics, University of Bergen, Norway
eiben@ac.tuwien.ac.at

Mithilesh Kumar
Department of Informatics, University of Bergen, Norway
mithilesh.kumar@ii.uib.no

Amer E. Mouawad
Department of Informatics, University of Bergen, Norway
a.mouawad@ii.uib.no

Fahad Panolan
Department of Informatics, University of Bergen, Norway
fahad.panolan@ii.uib.no

Sebastian Siebertz
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland
siebertz@mimuw.edu.pl

Abstract

For $\alpha > 1$, an α-approximate (bi-)kernel for a problem $\mathcal{Q}$ is a polynomial-time algorithm that takes as input an instance (I, k) of $\mathcal{Q}$ and outputs an instance (I', k') (of a problem $\mathcal{Q}'$) of size bounded by a function of k such that, for every $c \geq 1$, a c-approximate solution for the new instance can be turned into a $(c \cdot \alpha)$-approximate solution of the original instance in polynomial time. This framework of *lossy kernelization* was recently introduced by Lokshtanov et al. We study Connected Dominating Set (and its distance-r variant) parameterized by solution size on sparse graph classes like biclique-free graphs, classes of bounded expansion, and nowhere dense classes. We prove that for every $\alpha > 1$, Connected Dominating Set admits a polynomial-size α-approximate (bi-)kernel on all the aforementioned classes. Our results are in sharp contrast to the kernelization complexity of Connected Dominating Set, which is known to not admit a polynomial kernel even on 2-degenerate graphs and graphs of bounded expansion, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. We complement our results by the following conditional lower bound. We show that if a class $\mathcal{C}$ is somewhere dense and closed under taking subgraphs, then for some value of $r \in \mathbb{N}$ there cannot exist an α-approximate bi-kernel for the (Connected) Distance-r Dominating Set problem on $\mathcal{C}$ for any $\alpha > 1$ (assuming the Gap Exponential Time Hypothesis).

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis, Theory of computation → Parameterized complexity and exact algorithms, Theory of computation → Fixed parameter tractability, Theory of computation → Approximation algorithms analysis, Theory of computation → W hierarchy

Keywords and phrases Lossy Kernelization, Connected Dominating Set, Sparse Graph Classes

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.29

Funding Eduard Eiben was supported by Pareto-Optimal Parameterized Algorithms (ERC Starting Grant 715744) and by the Austrian Science Fund (FWF, projects P26696 and W1255-N23).

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 29; pp. 29:1–29:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The work of Sebastian Siebertz is supported by the National Science Centre of Poland via POLONEZ grant agreement UMO-2015/19/P/ST6/03998, which has received funding from the European Union's Horizon 2020 research and innovation programme (Marie Skłodowska-Curie grant agreement No. 665778).

1 Introduction

Lossy kernelization. A powerful method in parameterized complexity theory is to compute on input (I,k) a problem *kernel* in a polynomial-time pre-processing step, that is, to reduce the input instance in polynomial time to an equivalent instance (I',k') of size $g(k)$ for some function g bounded in the parameter only. If the reduced instance (I',k') belongs to a different problem than (I,k), we speak of a *bi-kernel*. It is well known that a problem is fixed-parameter tractable if and only if it admits a kernel, however, in general the function g can grow arbitrarily fast. For practical applications we are mainly interested in linear or at worst polynomial kernels. We refer to the textbooks [6, 11, 12] for extensive background on parameterized complexity and kernelization.

One shortcoming of the above notion of kernelization is that it does not combine well with approximation algorithms or heuristics. An approximate solution on the reduced instance provides no insight whatsoever about the original instance, the only statement we can derive from the definition of a kernel is that the reduced instance (I',k') is a positive instance if and only if the original instance (I,k) is a positive instance. This issue was recently addressed by Lokstanov et al. [23], who introduced the framework of *lossy kernelization*. Intuitively, the framework combines notions from approximation and kernelization algorithms to allow for approximation preserving kernels.

Formally, a *parameterized optimization (minimization or maximization) problem* Π over finite vocabulary Σ is a computable function $\Pi\colon \Sigma^\star \times \mathbb{N} \times \Sigma^\star \to \mathbb{R} \cup \{\pm\infty\}$. A *solution* for an instance $(I,k) \in \Sigma^\star \times \mathbb{N}$ is a string $s \in \Sigma^\star$, such that $|s| \leq |I| + k$. The *value* of the solution s is $\Pi(I,k,s)$. For a minimization problem, the *optimum value* of an instance (I,k) is $\mathrm{OPT}_\Pi(I,k) = \min_{s\in\Sigma^*,|s|\leq|I|+k} \Pi(I,k,s)$, for a maximization problem it is $\mathrm{OPT}_\Pi(I,k) = \max_{s\in\Sigma^*,|s|\leq|I|+k} \Pi(I,k,s)$. An *optimal solution* is a solution s with $\Pi(I,k,s) = \mathrm{OPT}_\Pi(I,k)$. If Π is clear from the context, we simply write $\mathrm{OPT}(I,k)$.

A vertex-subset graph problem $\mathcal{Q}$ defines which subsets of the vertices of an input graph are feasible solutions. We consider the following parameterized minimization problem associated with $\mathcal{Q}$:

$$\mathcal{Q}(G,k,S) = \begin{cases} \infty & \text{if } S \text{ is not a valid solution for } G \text{ as determined by } \mathcal{Q} \\ \min\{|S|, k+1\} & \text{otherwise.} \end{cases}$$

Note that this bounding of the objective function at $k+1$ does not make sense for approximation algorithms if one insists on k being the unknown optimum solution of the instance I. The parameterization above is by the value of the solution that we want our algorithms to output.

▶ **Definition 1.1.** Let $\alpha > 1$ and let Π be a parameterized minimization problem. An *α-approximate polynomial time pre-processing algorithm* $\mathcal{A}$ for Π is a pair of polynomial time algorithms. The first algorithm is called the *reduction algorithm*, and computes a map $R_\mathcal{A}\colon \Sigma^\star \times \mathbb{N} \to \Sigma^\star \times \mathbb{N}$. Given as input an instance (I,k) of Π, the reduction algorithm outputs another instance $(I',k') = R_\mathcal{A}(I,k)$. The second algorithm is called the *solution lifting algorithm*. It takes as input an instance $(I,k) \in \Sigma^\star \times \mathbb{N}$, the output instance

$(I', k') = R_{\mathcal{A}}(I, k)$, and a solution s' to the instance (I', k'). The solution lifting algorithm works in time polynomial in $|I|, k, |I'|, k'$ and s', and outputs a solution s to (I, k) such that

$$\frac{\Pi(I, k, s)}{\mathrm{OPT}(I, k)} \leq \alpha \cdot \frac{\Pi(I', k', s')}{\mathrm{OPT}(I', k')}.$$

▶ **Definition 1.2.** An *α-approximate kernelization algorithm* is an α-approximate polynomial time pre-processing algorithm for which we can prove an upper bound on the size of the output instances in terms of the parameter of the instance to be pre-processed. We speak of a linear or polynomial kernel, if the size bound is linear or polynomial, respectively. If we allow the reduced instance to be an instance of another problem, we speak of an *α-approximate bi-kernel*.

We refer to the work of Lokshtanov et al. [23] for an extensive discussion of related work and examples of problems that admit lossy kernels.

Sparse graphs and domination. We consider finite, undirected and simple graphs and refer to the textbook [10] for all undefined notation. We write $K_{i,j}$ for the complete bipartite graph with partitions of size i and j, respectively. We call a class $\mathcal{C}$ of graphs biclique-free if there are $i, j \in \mathbb{N}$ such that $K_{i,j}$ is not a subgraph of G for all $G \in \mathcal{C}$.

The notion of nowhere denseness was introduced by Nešetřil and Ossona de Mendez [28, 29] as a general model of *uniform sparseness* of graphs. Many familiar classes of sparse graphs, like planar graphs, graphs of bounded tree-width, graphs of bounded degree, and all classes that exclude a fixed (topological) minor, are nowhere dense. An important and related concept is the notion of a graph class of *bounded expansion*, which was also introduced by Nešetřil and Ossona de Mendez [25, 26, 27].

▶ **Definition 1.3.** Let H be a graph and let $r \in \mathbb{N}$. An *r-subdivision* of H is obtained by replacing all edges of H by internally vertex disjoint paths of length at most r.

▶ **Definition 1.4.** A class $\mathcal{C}$ of graphs is *nowhere dense* if there exists a function $t\colon \mathbb{N} \to \mathbb{N}$ such that for all $r \in \mathbb{N}$ and for all $G \in \mathcal{C}$ we do not find an r-subdivision of the complete graph $K_{t(r)}$ as a subgraph of G. Otherwise, $\mathcal{C}$ is called *somewhere dense*.

▶ **Definition 1.5.** A class $\mathcal{C}$ of graphs has *bounded expansion* if there exists a function $d\colon \mathbb{N} \to \mathbb{N}$ such that for all $r \in \mathbb{N}$ and all graphs H, such that an r-subdivision of H is a subgraph of G for some $G \in \mathcal{C}$, satisfy $|E(H)|/|V(H)| \leq d(r)$.

Every class of bounded expansion is nowhere dense, which in turn excludes some biclique as a subgraph and hence is biclique-free. For extensive background on bounded expansion and nowhere dense graphs we refer to the textbook of Nešetřil and Ossona de Mendez [30].

▶ **Definition 1.6.** In the parameterized Dominating Set (DS) problem we are given a graph G and $k \in \mathbb{N}$, and the objective is to determine the existence of a subset $D \subseteq V(G)$ of size at most k such that every vertex u of G is *dominated* by D, that is, either u belongs to D or has a neighbor in D. More generally, for fixed $r \in \mathbb{N}$, in the Distance-r Dominating Set (r-DS) problem we are asked to determine the existence of a subset $D \subseteq V(G)$ of size at most k such that every vertex $u \in V(G)$ is within distance at most r from a vertex of D. In the Connected (Distance-r) Dominating Set (CDS/r-CDS) problem we additionally demand that the (distance-r) dominating set shall be connected.

Dominating Set plays a central role in the theory of parameterized complexity, as it is a prime example of a W[2]-complete problem with the size of the optimal solution as the parameter, thus considered intractable in full generality. For this reason, the (Connected) Dominating Set problem and (Connected) Distance-r Dominating Set problem have been extensively studied on restricted graph classes. A particularly fruitful line of research in this area concerns kernelization algorithms for the aforementioned problems [1, 3, 15, 16, 17, 31]. Philip et al. [31] obtained a kernel of size $\mathcal{O}(k^{(d+1)^2})$ on d-degenerate graphs, for constant d, and more generally a kernel of size $\mathcal{O}(k^{\max(i^2,j^2)})$ on graphs excluding the biclique $K_{i,j}$ as a subgraph. On the lower bounds side, Cygan et al. [7] have shown that the existence of a size $\mathcal{O}(k^{(d-1)(d-3)-\epsilon})$ kernel, $\epsilon > 0$, for Dominating Set on d-degenerate graphs would imply NP $\subseteq$ coNP/poly. For the Connected Dominating Set problem linear kernels are only known for planar [22] and H-topological-minor-free graphs [17]. Polynomial kernels are excluded already for graphs of bounded degeneracy [8], assuming NP $\not\subseteq$ coNP/poly.

For the more general Distance-r Dominating Set problem we know the following results. Dawar and Kreutzer [9] showed that for every $r \in \mathbb{N}$ and every nowhere dense class $\mathcal{C}$, the Distance-r Dominating Set problem is fixed-parameter tractable on $\mathcal{C}$. Drange et al. [13] gave a linear bi-kernel for Distance-r Dominating Set on any graph class of bounded expansion for every $r \in \mathbb{N}$, and a pseudo-linear kernel for Dominating Set on any nowhere dense graph class; that is, a kernel of size $f(\epsilon) \cdot k^{1+\epsilon}$, for some function f and any $\epsilon > 0$. Precisely, the kernelization algorithm of Drange et al. [13] outputs an instance of an annotated problem where some vertices are not required to be dominated; this will be the case in the present paper as well (except for the case of biclique-free graphs). Kreutzer et al. [21] provided a polynomial bi-kernel for the Distance-r Dominating Set problem on every nowhere dense class for every fixed $r \in \mathbb{N}$ and finally, Eickmeyer et al. [14] could prove the existence of pseudo-linear bi-kernels of size $f(r, \epsilon) \cdot k^{1+\epsilon}$, for some function f.

It is known that bounded expansion classes of graphs are the limit for the existence of polynomial kernels for the Connected Dominating Set problem. Drange et al. [13] gave an example of a subgraph-closed class of bounded expansion which does not admit a polynomial kernel for Connected Dominating Set unless NP $\subseteq$ coNP/Poly. They also showed that nowhere dense classes are the limit for the fixed-parameter tractability of the Distance-r Dominating Set problem if we assume closure under taking subgraphs (classes which are closed under taking subgraphs will be called *monotone classes*).

Our results. In this paper we prove the following results.

- For any $\alpha > 1$, CDS admits an α-approximate kernel on $K_{d,d}$-free graphs of size $k^{\mathcal{O}(\frac{d^2}{\alpha-1})}$.
- For any $\alpha > 1$, CDS admits an α-approximate bi-kernel on graphs of bounded expansion of size $\mathcal{O}(f(\alpha) \cdot k)$ (i.e, linear in k), where $f(\alpha)$ is a function depending only on α.
- For every $\alpha > 1$ and every $r \in \mathbb{N}$, Connected Distance-r Dominating Set admits an α-approximate kernel of size polynomial in the parameter on classes of nowhere dense graphs, where the degree of the polynomial depends only on r and not on α.

Observe that our results are in sharp contrast with the above mentioned lower bounds on kernel size of traditional kernels on these classes. We complement our results by the following conditional lower bound. We show that if a class $\mathcal{C}$ is somewhere dense and closed under taking subgraphs, then for some value of $r \in \mathbb{N}$ there cannot exist an α-approximate bi-kernel for the (Connected) Distance-r Dominating Set problem on $\mathcal{C}$ for any $\alpha > 1$ (assuming the Gap Exponential Time Hypothesis).

Organization. We explain our methods and provide a general framework for computing lossy kernels for Connected Dominating Set in Section 2. Using this framework we can directly derive the claimed bounds for α-approximate kernels on biclique-free graphs, while for bounded expansion and nowhere dense classes we obtain α-approximate bi-kernels for Connected Distance-r Dominating Set of polynomial size with the degree of the polynomial depending both on r and on α. The rest of the paper is devoted to the far more technical part of improving the bounds for bounded expansion and nowhere dense classes of graphs. Due to space constraints, the conference version of this latter part contains only a proof outline.

2 A general framework

Although the technical details for dealing with biclique-free graphs and with bounded expansion and nowhere dense classes are quite different, the high-level approach is identical. The kernelization algorithms follow the same two-step strategy. First, our goal is to compute a "small" set of vertices whose domination is sufficient, i.e. the set of *dominatees* or the so-called domination core.

▶ **Definition 2.1** (k-domination core)**.** Let G be a graph and $Z \subseteq V(G)$. We say that Z is a *k-domination core* if every set D of size at most k that dominates Z also dominates $V(G)$.

Having found a domination core Z of appropriate size, the next step is to reduce the number of *dominators*, i.e. vertices whose role is to dominate other vertices, and the number of *connectors*, i.e. vertices whose role is to connect the solution.

▶ **Definition 2.2.** Let G be a graph and let $D, Z \subseteq V(G)$. We say that D is a *Z-dominator* if D dominates Z in G, i.e. every vertex $z \in Z \setminus D$ is at distance at most one from some vertex in D. We denote by $\mathbf{ds}(G, Z)$ ($\mathbf{cds}(G, Z)$) the size of a smallest (connected) Z-dominator in G. By $\mathbf{ds}(G)$ ($\mathbf{cds}(G)$) we mean $\mathbf{ds}(G, V(G))$ ($\mathbf{cds}(G, V(G))$).

We classify all vertices outside the core according to their domination properties.

▶ **Definition 2.3.** Let G be a graph and $Z \subseteq V(G)$. We define an equivalence relation $\sim_Z$ on $V(G) \setminus Z$ by $u \sim_Z v \Leftrightarrow N(u) \cap Z = N(v) \cap Z$.

Clearly, to find a kernel for Dominating Set it is now sufficient to construct the graph G' which contains the k-domination core Z and one representative of each equivalence class of $\sim_Z$. Then G admits a dominating set of size at most k if and only if G' contains a Z-dominator of size at most k. This simple two-step approach of computing a small domination core Z and then bounding the number of equivalence classes of the relation $\sim_Z$ forms the basis of the kernelization algorithms for Dominating Set in [13, 14, 21]. To control the number of classes of $\sim_Z$, we give the following definition. The *index* of an equivalence relation is the number of equivalence classes.

▶ **Definition 2.4.** Let G be a graph. We define the *neighborhood complexity function* of G as the function $\mu\colon \mathbb{N} \to \mathbb{N}$ with $\mu(z) = \max_{Z \subseteq V(G), |Z|=z} \textit{index}\ (\sim_Z)$.

For example all classes of bounded VC-dimension have polynomially bounded neighborhood complexity functions [33, 34], while bounded expansion classes have linear and nowhere dense classes have almost linear neighborhood complexity [18].

▶ **Proposition 2.5.** *Let $\mathcal{C}$ be a class of graphs such that the neighborhood complexity function for all $G \in \mathcal{C}$ is bounded by a fixed polynomial and such that on input (G, k), for $G \in \mathcal{C}$, we can decide in polynomial time whether $\mathbf{ds}(G) > k$ or otherwise compute a k-domination core $Z \subseteq V(G)$ of size polynomial in k. Then* Dominating Set *parameterized by k admits a polynomial-size kernel on $\mathcal{C}$.*

The above proposition can be applied, e.g., to obtain a polynomial kernel on biclique-free graphs (we will prove the existence of a polynomial k-domination core below). However, as the hardness results even for degenerate graphs show, this approach does not extend to connected dominating sets. We may have to include more vertices in the kernel to ensure connectivity of the dominating sets. This turns out to be a major problem for the construction of polynomial size kernels for Connected Dominating Set.

When reducing the number of vertices outside the domination core, we borrow approximation techniques that are closely related to the Steiner Tree problem.

▶ **Definition 2.6.** Let G be a graph and let $Y \subseteq V(G)$ be a set of *terminals*. A *Steiner tree* for Y is a subtree of G spanning Y. We write $\mathbf{st}_G(Y)$ for the order of (i.e. the number of vertices of) the smallest Steiner tree for Y in G (including the vertices of Y). If $\mathcal{Y} = \{V_1, \ldots, V_t\}$ is a family of vertex disjoint subsets of G, a *group Steiner tree* for $\mathcal{Y}$ is a subtree of G that contains (at least) one vertex of each group V_i. We write $\mathbf{st}_G(\mathcal{Y})$ for the order of the smallest group Steiner tree for $\mathcal{Y}$.

The Group Steiner Tree problem on t groups can be solved in $\mathcal{O}(2^t \cdot n^{\mathcal{O}(1)})$-time [24].

The following definition and proposition form the key to our approach to handle connectivity in the lossy kernelization framework.

▶ **Definition 2.7.** Let D be a connected graph and $t \in \mathbb{N}$. A (D, t)-*covering family* is a family $\mathcal{F}(D, t)$ of connected subgraphs of D such that (i) for each $T \in \mathcal{F}(D, t)$, $|V(T)| \leq 2t$ and (ii) $\bigcup_{T \in \mathcal{F}(D,t)} V(T) = V(D)$.

▶ **Proposition 2.8.** *Let D be a connected graph and $t \in \mathbb{N}$. Then there is a (D, t)-covering family $\mathcal{F}(D, t)$ such that $|\mathcal{F}(D, t)| \leq \frac{|V(D)|}{t} + 1$, and $\sum_{T \in \mathcal{F}(D,t)} |V(T)| \leq (1 + \frac{1}{t})|V(D)| + 1$.*

Proof Sketch. Let T_D be a spanning tree of D. We create a (D, t)-covering family $\mathcal{F}(D, t) = \{T_1, T_2, \ldots, T_\ell\}$, which is a set of subtrees of T_D constructed as follows. We root T_D at an arbitrary vertex $r \in V(T_D)$. For any pair of vertices $u, v \in V(T_D)$, u is called a child of v if $uv \in E(T_D)$ and v lies on the path from u to r. For each vertex $v \in V(T_D)$, we let $\text{weight}(v) = 1 + \sum_{u \in \text{child}(v)} \text{weight}(u)$, where $\text{child}(v)$ denotes the set of children of v in T_D. In other words, $\text{weight}(v)$ is the number of vertices in the subtree rooted at v. Leaves have weight one. We use T_v to denote the subtree rooted at v. We construct $\mathcal{F}(D, t) = \{T_1, T_2, \ldots\}$ from T_D as follows:

1. If T_D is empty, terminate. Otherwise, compute the weights of all vertices in T_D, then sort the vertices in increasing order of weight.
2. If there exists a vertex whose weight is between t and $2t$ (inclusive), pick the vertex with the smallest such weight, add T_v to $\mathcal{F}(D, t)$, delete T_v from T_D, then go back to step (1).
3. If there exists a vertex whose weight is strictly greater than $2t$, pick the vertex with the smallest such weight, greedily compute a subset $S \subseteq \text{child}(v)$ such that $t < \sum_{u \in S} \text{weight}(u) < 2t$, let $R = \text{child}(v) \setminus S$, add $T_v - \bigcup_{w \in R} V(T_w)$ to $\mathcal{F}(D, t)$, delete T_u from T_D, for every $u \in S$, then go back to step (1). Note that by our choice of v, all children of v must have weight at most $t - 1$ as otherwise case (2) would apply.

4. Otherwise, every vertex in T_D has weight strictly less than t and hence T_D has at most t vertices (by the definition of the weight function). In this case, simply add T_D to $\mathcal{F}(D,t)$ and terminate.

The correctness proof is deferred to the full version of the paper due to the paucity of space. ◀

While the previous proposition allows us to "take apart" connected dominating sets, the next proposition explains how to "put them back together".

▶ **Proposition 2.9.** *Let G be a graph, $X \subseteq V(G)$, such that $G[X]$ is connected, and let D be an X-dominator such that $G[D]$ has at most p connected components. Then a set $Q \subseteq X$ of size at most $2p$ such that $G[D \cup Q]$ is connected, can be computed in polynomial time.*

We have now collected all the tools required to control the number of vertices which have to be added to ensure connectivity.

▶ **Theorem 2.10.** *Let $\mathcal{C}$ be a class of graphs such that the neighborhood complexity function for all $G \in \mathcal{C}$ is bounded by a fixed polynomial of degree d and such that on input (G,k), for $G \in \mathcal{C}$, we can decide in polynomial time whether $\mathbf{cds}(G) > k$ or otherwise compute a k-domination core $Z \subseteq V(G)$. Then for every $\epsilon > 0$,* Connected Dominating Set *parameterized by k admits a $(1+\epsilon)$-approximate kernel with $|Z|^{\mathcal{O}(d/\epsilon)}$ vertices on $\mathcal{C}$.*

The reduction algorithm. Let (G,k) be the input instance, where $G \in \mathcal{C}$ is connected and k is a positive integer. We first describe the reduction algorithm $\mathcal{R}_{\mathcal{A}}$. As a first step we run the polynomial time algorithm (which exists by assumption of the theorem) to decide whether $\mathbf{cds}(G) > k$ and otherwise compute a k-domination core $Z \subseteq V(G)$. In the first case, we output a trivial negative instance $((\{v\},\emptyset),0)$. In the second case, we proceed as follows.

We partition the graph into two sets Z and $R = V(G) \setminus Z$. We compute the equivalence relation $\sim_Z$ on R (see Definition 2.3), that is, we partition vertices in R according to their neighborhoods in Z. This is clearly possible in polynomial time. Let $\mathcal{R}$ be the set of equivalence classes defined by $\sim_Z$. As a direct implication of our assumption, we can bound the size of $\mathcal{R}$ by $\mathcal{O}(|Z|^d)$.

▶ **Proposition 2.11.** *The equivalence relation $\sim_Z$ has $\mathcal{O}(|Z|^d)$ classes, i.e. $|\mathcal{R}| \in \mathcal{O}(|Z|^d)$.*

As Z is a k-domination core, to find a dominating set of size at most k it is enough to find a set which dominates Z. Hence for the purpose of domination, it is redundant to pick more than one vertex from an equivalence class in $\mathcal{R}$. The following construction finds a small set of relevant vertices which "approximately" preserves the connectivity requirements.

Let $t \geq 1$ be a constant, which we fix later. Let $\mathcal{Z}$ be the family of groups $\{\{z\} \mid z \in Z\}$ and let $\mathcal{R}$ be the set of equivalence classes defined by $\sim_Z$. The set $\mathcal{R} \cup \mathcal{Z}$ forms a family of groups of vertices in $V(G)$. For every subset $\mathcal{Q} = \{Q_1, \ldots, Q_\ell\} \subseteq \mathcal{R} \cup \mathcal{Z}$ of size at most $2t$ of groups in $\mathcal{R} \cup \mathcal{Z}$, construct a Group Steiner Tree instance on the graph G with groups $Q_1, \ldots, Q_\ell$. Note that since t is a constant each instance can be solved in polynomial time using the algorithm of Misra et al. [24]. For each subset $\mathcal{Q}$ denote by $T_{\mathcal{Q}}$ the corresponding solution. For every instance that we solve, if the size of $T_{\mathcal{Q}}$ is at most $2t$ then we mark the vertices of $T_{\mathcal{Q}}$ in G. We denote the set of all marked vertices by $\bigcup T_{\mathcal{Q}}$. If $\bigcup T_{\mathcal{Q}}$ is not a dominating set in G, then we may declare that $\mathbf{cds}(G) > k$. Otherwise, since G is assumed to be connected, we can run the polynomial-time algorithm of Proposition 2.9 (with parameter $X = V(G)$) to obtain a set $W \subseteq V(G)$ such that $\bigcup T_{\mathcal{Q}} \cup W$ is a connected dominating set in G and $|\bigcup T_{\mathcal{Q}} \cup W| \leq 3|\bigcup T_{\mathcal{Q}}|$. Let $Y = \bigcup T_{\mathcal{Q}} \cup W$. We output the instance $(G[Y],k)$.

Approximation guarantee. Now we prove that $\mathrm{OPT}(G[Y],k) \leq (1+\epsilon)\mathrm{OPT}(G,k)$. Let D^* be a connected dominating set of G of minimum cardinality. If $|D^*| > k$, then $\mathrm{OPT}(G[Y],k) \leq (1+\epsilon)\mathrm{OPT}(G,k)$ holds trivially. So we assume that $|D^*| \leq k$. We let $\mathcal{F}(D^*,t) = \{T_1, T_2, \cdots, T_m\}$ denote a (D^*,t)-covering family. Proposition 2.8 implies that there exists such a family for which $|\mathcal{F}(D^*,t)| \leq \frac{|V(D^*)|}{t}+1$ and $\sum_{T\in\mathcal{F}(D^*,t)} |V(T)| \leq (1+\frac{1}{t})|V(D^*)|+1$. Moreover, the size of each connected subgraph T (in this case also subtree) is at most $2t$. We construct a new family $\mathcal{F}'$ from $\mathcal{F}(D^*,t)$ as follows. For each $T \in \mathcal{F}(D^*,t)$, we replace T by $T_{\mathcal{Q}}$, where $\mathcal{Q}$ is the set of groups from $\mathcal{R}\cup\mathcal{Z}$ such that $Q \in \mathcal{Q}$ if and only if $V(T)\cap Q \neq \emptyset$ and $T_{\mathcal{Q}}$ is the set of marked vertices in an optimal Steiner tree connecting vertices from the groups in $\mathcal{Q}$. Note that the fact that T is of size at most $2t$ guarantees the existence of $T_{\mathcal{Q}}$ (by construction). Moreover, the size of $T_{\mathcal{Q}}$ is at most the size of T, since T is also a solution for Group Steiner Tree for $\mathcal{Q}$. Let $D_{\mathcal{F}'}$ denote the union of all vertices in $\mathcal{F}'$.

Let D' be a subset of $D_{\mathcal{F}'}$, of cardinality at most $|D^*|$, such that for any $w \in D^*$, there is a vertex $w' \in D'$ with the property that $\{w,w'\} \subseteq Q \in \mathcal{R}\cup\mathcal{Z}$ and $w' \in D_{\mathcal{F}'}$. That is, if D^* has a vertex from a group Q in $\mathcal{R}\cup\mathcal{Z}$, then D' also has a vertex from group Q. We claim that D' is a dominating set in G. Notice that $Z\cap V(D) = Z\cap D'$ and if any vertex in Z is adjacent to a vertex in a group Q, then it is adjacent to all vertices in group Q. This implies that D' also dominates Z and since $|D'| \leq |D| \leq k$, by the definition of a k-domination core, D' is a dominating set in G.

This implies that $D_{\mathcal{F}'} \supseteq D'$ is also a dominating set in G. Applying Proposition 2.9 in $G[Y]$ (with $D_{\mathcal{F}'}$ as dominator and since $G[Y]$ is connected), we obtain a connected dominating set of size at most $2|\mathcal{F}(D^*,t)| + |D_{\mathcal{F}'}| \leq \frac{2|V(D^*)|}{t} + 2 + (1+\frac{1}{t})|V(D^*)| + 1 = (1+\frac{3}{t})|V(D^*)| + 3$. Now we can fix the constant t appropriately (as roughly $\frac{3}{\epsilon}$) and we get that $\mathrm{OPT}(G[Y],k) \leq (1+\epsilon)\mathrm{OPT}(G,k)$.

Size of the kernel. Now we show that $|Y| \in |Z|^{\mathcal{O}(d)/\epsilon}$. By Proposition 2.11, we have that $|\mathcal{R}\cup\mathcal{Z}| = \mathcal{O}(Z^d)$. From the construction, it follows that $|\bigcup T_{\mathcal{Q}}| = \mathcal{O}(2t|\mathcal{R}\cup\mathcal{Z}|^{\mathcal{O}(t)}) = |Z|^{\mathcal{O}(d/\epsilon)}$. Notice that $Y = \bigcup T_{\mathcal{Q}} \cup W$, where W is obtained by applying Proposition 2.9 and hence we have that $Y = |\bigcup T_{\mathcal{Q}} \cup W| \leq 3|\bigcup T_{\mathcal{Q}}| = |Z|^{\mathcal{O}(d/\epsilon)}$.

The solution lifting algorithm. The solution lifting algorithm works as follows. Given a solution D' to the reduced instance (G',k'), if D' is not a connected dominating set of G', then the solution lifting algorithm will output $\emptyset$. If D' is a connected dominating set, then the algorithm returns D' if $|D'| \leq k$ and $V(G)$ otherwise. Let D be the output of the solution lifting algorithm.

The final step. We prove that the above reduction algorithm together with the solution lifting algorithm constitute a $(1+\epsilon)$-approximate kernel. Note that if D' is not a valid solution of G', then $\emptyset$ is not a valid solution for G and $\mathrm{CDS}(G',k',D') = \mathrm{CDS}(G,k,D) = \infty$. Hence we can restrict ourselves to the case when D' is a connected dominating set of G'. First, consider the case where the reduction algorithm outputs $Y \subseteq V(G)$ and the reduced instance is hence $(G',k') = (G[Y],k)$. From our above observation, we have that $\mathrm{OPT}(G[Y],k) \leq (1+\epsilon)\mathrm{OPT}(G,k)$. We show that in this case $\mathrm{CDS}(G,k,D) = \mathrm{CDS}(G',k',D')$. If $|D'| > k$, then $\mathrm{CDS}(G,k,D) = \mathrm{CDS}(G,k,V(G)) = k+1 = \mathrm{CDS}(G',k',D')$. So assume that $|D'| \leq k$, which implies $D = D'$. Since D' is a connected dominating set of $G[Y]$ and Y contains a k-domination core of G, it follows that D' dominates G and $\mathrm{CDS}(G,k,D) = \mathrm{CDS}(G',k',D')$. Combining $\mathrm{CDS}(G,k,D) = \mathrm{CDS}(G',k',D')$ and $\mathrm{OPT}(G[Y],k) \leq (1+\epsilon)\mathrm{OPT}(G,k)$ we get $\frac{\mathrm{CDS}(G,k,D)}{\mathrm{OPT}(G,k)} \leq (1+\epsilon)\frac{\mathrm{CDS}(G',k',D')}{\mathrm{OPT}(G',k')}$.

When $(G', k') = ((\{v\}, \emptyset), 0)$, we can easily verify that the above mentioned approximation guarantee holds. □

The remainder of the paper is concerned with proving the existence of small domination cores for concrete sparse classes of graphs. For example, to prove the following theorem for biclique-free graphs we prove the existence of a polynomial k-domination core and simply use the fact that such classes have polynomially bounded neighborhood complexity.

▶ **Theorem 2.12.** *For every $\epsilon > 0$,* Connected Dominating Set*, parameterized by solution size, admits a $(1+\epsilon)$-approximate kernel with $k^{\mathcal{O}(d^2/\epsilon)}$ vertices on $K_{d,d}$-free graphs.*

The most technical part will be to prove the existence of a linear domination core for bounded expansion classes (the definition of a domination core is slightly changed to obtain such good bounds, see Section 3). Most surprisingly, the general framework summarized in Theorem 2.10 does not produce a bi-kernel of size $\mathcal{O}(|Z|^{1/\epsilon})$ but rather of size $f(\epsilon) \cdot |Z|$ for some function f on bounded expansion classes and of polynomial size on nowhere dense classes.

3 Graphs of bounded expansion

In this section we show that Connected Dominating Set, parameterized by solution size k, admits a $(1+\epsilon)$-approximate bi-kernel on at most $\mathcal{O}(f(\epsilon) \cdot k)$ vertices on graphs of bounded expansion. The reduced instance will be an instance of Subset Connected Dominating Set (SCDS), defined as follows:

$$\mathrm{SCDS}((G, Z), k, D) = \begin{cases} \infty & \text{if } D \text{ is not a connected } Z\text{-dominator in } G \\ \min\{|D|, k+1\} & \text{otherwise} \end{cases}$$

We first formally define the class of graphs of bounded expansion. Towards that we need the definition of shallow minors.

▶ **Definition 3.1.** A graph M is an *r-shallow minor* of G, for some $r \in \mathbb{N}$, if there is a family of disjoint subsets $V_1, \ldots, V_{|M|}$ of $V(G)$ such that (i) each graph $G[V_i]$ is connected and has radius at most r, and (ii) there is a bijection $\omega \colon V(M) \to \{V_1, \ldots, V_{|M|}\}$ such that for any $uv \in E(M)$ there is an edge in G with one endpoint in $\omega(u)$ and another in $\omega(v)$.

The set of all r-shallow minors of a graph G is denoted by $G\nabla r$. The set of all r-shallow minors of all members of a graph class $\mathcal{G}$ is denoted by $\mathcal{G}\nabla r = \bigcup_{G \in \mathcal{G}} (G\nabla r)$. It will be convenient to work with the following equivalent definition of bounded expansion classes.

▶ **Definition 3.2** (Grad and bounded expansion [25])**.** For a graph G and an integer $r \geq 0$, the *greatest reduced average density (grad) at depth r* is, $\nabla_r(G) = \max_{M \in G\nabla r} \mathrm{density}(M) = \max_{M \in G\nabla r} |E(M)|/|V(M)|$. For a graph class $\mathcal{G}$, $\nabla_r(\mathcal{G}) = \sup_{G \in \mathcal{G}} \nabla_r(G)$. A graph class $\mathcal{G}$ has *bounded expansion* if there is a function $f \colon \mathbb{N} \to \mathbb{R}$ such that for all r we have $\nabla_r(\mathcal{G}) \leq f(r)$.

The first phase of our algorithm, i.e. finding a domination core, closely follows the work of Drange et al. [13] but requires subtle changes.

3.1 Finding the domination core

In the following we fix a graph class $\mathcal{G}$ that has bounded expansion and let (G, k) be the input instance of Connected Dominating Set, where $G \in \mathcal{G}$ and G is connected.

To obtain a linear domination core for classes of bounded expansion we have to invest a considerable amount of work. The following construction shows that we cannot work with k-domination cores.

▶ **Lemma 3.3.** *There exists a class $\mathcal{C}$ of bounded expansion such that for all $k \in \mathbb{N}$ there is $G \in \mathcal{C}$ such that every k-domination core for G has $\Omega(k^2)$ vertices.*

Instead, we introduce the notion of a c-exchange domination core, which is different from the definition used in the previous section and from the one considered in [13]. Here, c is a fixed constant which we set later.

▶ **Definition 3.4** (c-exchange domination core)**.** Let G be a graph and $Z \subseteq V(G)$. We say that Z is a *c-exchange domination core* if for every set X that dominates Z one of the following conditions holds: (1.) X dominates G, or (2.) there exist $A \subseteq X$ and $B \subseteq V(G)$ such that $|B| < |A| \leq c$ and $(X \setminus A) \cup B$ dominates Z. Moreover the number of connected components of $(X \setminus A) \cup B$ is at most the number of connected components of X. In particular, if X is a connected set then $(X \setminus A) \cup B$ is also connected.

Clearly, $V(G)$ is a c-exchange domination core, for any c, but we look for a c-exchange domination core that is linear in k. Hence, we start with $Z = V(G)$ and gradually reduce $|Z|$ by removing one vertex at a time, while maintaining the invariant that Z is a c-exchange domination core. To this end, we need to prove Lemma 3.5. Note that we only remove vertices from Z at this stage (no vertex deletions), and hence the graph remains intact.

▶ **Lemma 3.5.** *There exists a constant $C_{\textbf{\textit{core}}} > 0$ depending only on a fixed (finite) number of grads of $\mathcal{G}$ and a polynomial-time algorithm that, given a graph $G \in \mathcal{G}$ and a c-exchange domination core $Z \subseteq V(G)$ with $|Z| > C_{\textbf{\textit{core}}} \cdot k$, either correctly concludes that $\mathbf{cds}(G) > k$ or finds a vertex $z \in Z$ such that $Z \setminus \{z\}$ is still a c-exchange domination core. Here c is a non-zero positive constant.*

We remark that we do not know how to prove the existence of a *small* c-exchange distance-r domination core (a generalization needed for r-CDS) on graphs of bounded expansion. In fact the generalization of our proof will not guarantee that the number of connected components after exchange is at most the number of connected component before exchange. This obstacle prevents us from finding a linear or almost linear bi-kernel for the case of r-CDS.

3.2 Reducing connectors and dominators

Armed with a c-exchange domination core Z whose size is linear in k, our next goal is to reduce the number of connectors and dominators (the number of vertices in $V(G) \setminus Z$). To that end, we need the following lemma which is a generalized version of Lemma 2.11 in [13].

▶ **Lemma 3.6** (Trees closure lemma)**.** *Let $\mathcal{G}$ be a class of bounded expansion and let q and r be positive integers. Let $G \in \mathcal{G}$ be a graph and $X \subseteq V(G)$. Then a superset of vertices $X' \supseteq X$ can be computed in polynomial time, with the following properties: (1) For every $Y \subseteq X$ of size at most q, if $\mathbf{st}_G(Y) \leq rq$ then $\mathbf{st}_{G[X']}(Y) = \boldsymbol{st}_G(Y)$. (2) $|X'| \leq C_{\boldsymbol{tc}} \cdot |X|$, where $C_{\boldsymbol{tc}}$ is a constant depending only on r, q, and the class $\mathcal{G}$.*

Lemma 3.6 ensures that the reduction algorithm of Theorem 2.10 produces a set $\bigcup T_Q$ of linear size. We addionally use the linear neighborhood complexity of classes of bounded expansion to conclude as in the proof of Theorem 2.10.

The rest of section is devoted to prove Lemma 3.6. Towards that we need some more definitions and known results which we state first. Given two graphs G and H, the *lexicographic product* $G \odot H$ is defined as the graph on the vertex set $V(G) \times V(H)$ where vertices (u, a) and (v, b) are adjacent if $uv \in E(G)$ or if $u = v$ and $ab \in E(H)$.

▶ **Lemma 3.7** ([19, 20]). *For a graph G and non-negative integers $t \geq 1$ and r we have $\nabla_r(G \odot K_t) \leq 5t^2(r+1)^2 \cdot \nabla_r(G)$.*

Let G be a graph and X be a subset of its vertices. For $u \in V(G) \setminus X$, we define the *r-projection* of u onto X as follows: $M_r^G(u, X)$ is the set of all vertices $w \in X$ for which there exists a path in G that starts in u, ends in w, has length at most r, and whose internal vertices do not belong to X. Note that $M_1^G(u, X) = N_X(u)$.

▶ **Lemma 3.8** ([13]). *Let $\mathcal{G}$ be a class of graphs of bounded expansion. There exists a polynomial-time algorithm that, given a graph $G \in \mathcal{G}$, $X \subseteq V(G)$, and an integer $r \geq 1$, computes the r-*closure *of X, denoted by $cl_r(X)$, with the following properties:*

(i) $X \subseteq cl_r(X) \subseteq V(G)$,

(ii) $|cl_r(X)| \leq C_{\mathbf{cl1}} \cdot |X|$, and

(iii) $|M_r^G(u, cl_r(X))| \leq C_{\mathbf{cl2}}$ for each $u \in V(G) \setminus cl_r(X)$, where $C_{\mathbf{cl1}}$ and $C_{\mathbf{cl2}}$ are constants depending only on r and a fixed (finite) number of grads of $\mathcal{G}$.

Proof of Lemma 3.6. First, using Lemma 3.8 we compute $X_0 = \mathrm{cl}_{rq}(X)$. Then, $|X_0| \leq C_{\mathbf{cl1}} \cdot |X|$ and for each vertex $u \notin X_0$ we have $|M_{rq}^G(u, X_0)| \leq C_{\mathbf{cl2}}$. Now, for each set $Y \subseteq X_0$ of at most q vertices, compute an optimal Steiner tree T_Y whose edges do not belong to $G[X_0]$; in case there is no such tree, set $T_Y = \emptyset$. Note that T_Y can be computed in polynomial time for any fixed q [2]. Define X' to be X_0 plus the vertex sets of all trees T_Y that have size at most rq.

▶ **Claim 3.9.** *$|X'| \leq C_{\mathbf{tc}} \cdot |X_0|$, where $C_{\mathbf{tc}}$ is a constant depending only on r, q, and a finite number of grads of $\mathcal{G}$.*

Proof of the Claim. Let H be a graph on vertex set X_0, where $uv \in E(H)$ if and only if there exists Y such that $\{u, v\} \subseteq Y$, $T_Y \neq \emptyset$ and has size at most rq, and hence its vertex set was added to X. Note that we do not add multiedges. For every such set Y, $H[Y]$ induces a clique in H. Let $\omega(H)$ denote the number of cliques in H. Clearly $|X'| \leq |X_0| + rq \cdot \omega(H)$, so it suffices to prove an upper bound on $\omega(H)$.

Consider an edge $uv \in E(H)$. The existence of this edge implies that u and v appear together in some tree T_Y of size at most rq. Since T_Y does not contain any edges from $G[X_0]$ (by construction), there must exist a path $P_{u,v}$ of length at most rq connecting u and v. The internal vertices of $P_{u,v}$ do not belong to X_0. Take any $w \in X' \setminus X_0$, and consider for how many pairs $\{u, v\} \subseteq X_0$ it can hold that $w \in P_{u,v}$. If $\{u, v\}$ is such a pair, then in particular $u, v \in M_{rq}^G(w, X_0)$. But we know that $|M_{rq}^G(w, X_0)| \leq C_{\mathbf{cl2}}$, so the number of such pairs is at most $\tau \leq (C_{\mathbf{cl2}})^2$. Consequently, we observe that graph H is an $(rq-1)$-shallow minor of $G \odot K_\tau$: when each vertex $w \in X' \setminus X_0$ is replaced with τ copies, then we can realize all the paths between u and v, in $G \odot K_\tau$, so that they are internally vertex-disjoint. From Lemma 3.7, we know that $\nabla_{rq-1}(G \odot K_\tau)$ is bounded polynomially in $\nabla_{rq-1}(G)$ and τ, which in turn is also bounded polynomially in $\nabla_{rq-1}(\mathcal{G})$. Hence $\nabla_{rq-1}(G \odot K_\tau)$ is bounded polynomially in $\nabla_{rq-1}(\mathcal{G})$. The number of cliques in graph of bounded expansion is linear in the number of vertices [4]. Combining the fact that H has bounded expansion with $|X'| \leq |X_0| + rq \cdot \omega(H)$, the claim follows. ◀

▶ **Claim 3.10.** *If $Y \subseteq X_0$ has size at most q and $\mathbf{st}_G(Y) \leq rq$ then $\mathbf{st}_{G[X']}(Y) = \boldsymbol{st}_G(Y)$.*

Proof of the Claim. Let T_Y be an optimal Steiner tree for Y in G, and let $T_1, T_2, \ldots, T_p$ be the subtrees of size greater than one obtained after deleting all edges of T_Y for which both endpoints are in X_0. Note that deleting such edges can only create either singleton vertices or subtrees of size greater than one. Moreover, let Y_i, $1 \leq i \leq p$, denote the set $Y \cap V(T_i)$. The existence of T_i certifies that some tree of size at most $|T_i|$ was added when constructing X' from X_0, and hence $\mathbf{st}_{G[X']}(Y_i) \leq |T_i|$. Consequently, we infer that

$$\mathbf{st}_{G[X']}(Y) \leq \sum_{i=1}^{p} \mathbf{st}_{G[X']}(Y_i) + |Y \setminus \bigcup_{i=1}^{p} Y_i| \leq \sum_{i=1}^{p} |T_i| + |Y \setminus \bigcup_{i=1}^{p} Y_i| \leq |T_Y| = \mathbf{st}_G(Y).$$

The opposite inequality $\mathbf{st}_{G[X']}(Y) \geq \mathbf{st}_G(Y)$ follows directly from the fact that $G[X']$ is an induced subgraph of G. ◀

Claim 3.9 and the fact that $|X_0| \leq C_{\mathbf{cl1}}|X|$ prove property (2). Claim 3.10 and the fact that $X \subseteq X_0$ prove property (1). ◀

4 Nowhere dense graphs

As we solve the more general Connected Distance-r Dominating Set on nowhere dense classes, we work with the following definition of a domination core.

▶ **Definition 4.1.** Let G be a graph. A set $Z \subseteq V(G)$ is a (k, r)-*domination core* for G if every set D of size at most k that r-dominates Z also r-dominates G

Domination cores of polynomial size exist for nowhere dense classes, as the following lemma shows.

▶ **Lemma 4.2** (Kreutzer et al. [21])**.** *There exists a polynomial q (of degree depending only on r) and a polynomial-time algorithm that, given a graph $G \in \mathcal{C}$ and $k \in \mathbb{N}$ either correctly concludes that G cannot be r-dominated by a set of at most k vertices, or finds a (k, r)-domination core $Z \subseteq V(G)$ of G of size at most $q(k)$.*

We remark that the non-constructive polynomial bounds that follow from [21] can be replaced by the much improved constructive bounds from [32].

It now remains to prove a lemma analogous to Lemma 3.6 for nowhere dense classes to conclude again similarly as in the proof of Theorem 2.10.

5 Lower bounds

Our lower bound is based on Proposition 3.2 of [23] which establishes equivalence between FPT-approximation algorithms and approximate kernelization. More precisely, the Proposition states that for every function α and decidable parameterized optimization problem Π, Π admits a fixed parameter tractable α-approximation algorithm if and only if Π has an α-approximate kernel.

We use a reduction from Set Cover to the Distance-r Dominating Set problem, which under the assumption that the Gap Exponential Time Hypothesis (gap-ETH) holds does not admit a fixed-parameter tractable α-approximation algorithm for any function α [5].

For every monotone somewhere dense graph class $\mathcal{C}$, there exists $r \in \mathbb{N}$ such that the exact r-subdivision of every graph can be found as a member of $\mathcal{C}$ [29]. Now it is straight forward to adapt the W[2]-hardness proof for Distance-r Dominating Set for monotone somewhere dense graph classes [13] to our setting.

▶ **Theorem 5.1.** *If the Gap Exponential Time Hypothesis holds, then for every monotone somewhere dense class of graphs $\mathcal{C}$ there is no $\alpha(k)$-approximate kernel for the* DISTANCE-r DOMINATING SET *problem on $\mathcal{C}$ for any function $\alpha\colon \mathbb{N} \to \mathbb{N}$.*

6 Conclusion

The study of computationally hard problems on restricted classes of inputs is a very fruitful line of research in algorithmic graph structure theory and in particular in parameterized complexity theory. This research is based on the observation that many problems such as DOMINATING SET, which are considered intractable in general, can be solved efficiently on restricted graph classes. In this work we were able to provide lossy kernels for graphs of bounded expansion whose size matches the size of the best known kernel for DOMINATING SET. We were furthermore able to identify the exact limit for the existence of lossy kernels for r-CDS. One interesting open question is whether our polynomial bounds on the size of the lossy kernel on nowhere dense classes can be improved to pseudo-linear bounds. For $K_{d,d}$-free graphs we have an additional $\frac{1}{\alpha-1}$ multiplicative factor in the exponent. This leads to the question whether it is possible to reduce the size of our kernel on $K_{d,d}$-free graphs to $f(\alpha)k^{\mathcal{O}(d^2)}$ for some function f. And, in light of the $\mathcal{O}(k^{(d-1)(d-3)-\epsilon})$ lower bound for DOMINATING SET, is it possible to obtain a lossy kernel for DOMINATING SET on biclique-free graphs that beats this bound? Our hope is that such a "fine-grained" analysis of the kernelization complexity of domination problems will lead to a better understanding of the boundary between "hard" and "easy" instances.

References

1. Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, 2004.
2. Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: Fast subset convolution. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, pages 67–74, New York, NY, USA, 2007.
3. Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. In *50th Annual IEEE Symposium on Foundations of Computer Science, Atlanta, Georgia, USA*, pages 629–638, 2009.
4. Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.
5. Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-eth to fpt-inapproximability: Clique, dominating set, and more. *arXiv preprint arXiv:1708.04218*, 2017.
6. Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 3. Springer, 2015.
7. Marek Cygan, Fabrizio Grandoni, and Danny Hermelin. Tight kernel bounds for problems on graphs with small degeneracy - (extended abstract). In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 361–372, 2013.
8. Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Kernelization hardness of connectivity problems in d-degenerate graphs. *Discrete Applied Mathematics*, 160(15):2131–2141, 2012.

9 Anuj Dawar and Stephan Kreutzer. Domination problems in nowhere-dense classes. In *FSTTCS 2009*, volume 4 of *LIPIcs*, pages 157–168. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2009.

10 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

11 Rodney G Downey and Michael R Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.

12 Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer, 1999.

13 Pål Grønås Drange, Markus Sortland Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, Felix Reidl, Fernando Sánchez Villaamil, Saket Saurabh, Sebastian Siebertz, and Somnath Sikdar. Kernelization and sparseness: the case of dominating set. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016*, pages 31:1–31:14, 2016.

14 Kord Eickmeyer, Archontia C. Giannopoulou, Stephan Kreutzer, O-joung Kwon, Michal Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. Neighborhood complexity and kernelization for nowhere dense classes of graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 63:1–63:14, 2017.

15 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, Austin, Texas, USA*, pages 503–510, 2010.

16 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Linear kernels for (connected) dominating set on H-minor-free graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 82–93, 2012.

17 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Linear kernels for (connected) dominating set on graphs with excluded topological subgraphs. In *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, pages 92–103, 2013.

18 Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. *J. Comput. Syst. Sci.*, 84:219–242, 2017. doi:10.1016/j.jcss.2016.09.002.

19 Sariel Har-Peled and Kent Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs, 2015. (CoRR/abs/1501.00721). URL: http://arxiv.org/abs/1501.00721.

20 Sariel Har-Peled and Kent Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 717–728. Springer, 2015.

21 Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Polynomial kernels and wideness properties of nowhere dense graph classes. In *SODA*, pages 1533–1545. SIAM, 2017.

22 Daniel Lokshtanov, Matthias Mnich, and Saket Saurabh. A linear kernel for a planar connected dominating set. *Theor. Comput. Sci.*, 412(23):2536–2543, 2011.

23 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 224–237, 2017.

24 Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. FPT algorithms for connected feedback vertex set. In *WALCOM: Algorithms and*

Computation, 4th International Workshop, WALCOM 2010, Dhaka, Bangladesh, February 10-12, 2010. Proceedings, pages 269–280, 2010.

25 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion I. Decompositions. *European Journal of Combinatorics*, 29(3):760–776, 2008.

26 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion II. Algorithmic aspects. *European Journal of Combinatorics*, 29(3):777–791, 2008.

27 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion III. Restricted graph homomorphism dualities. *European Journal of Combinatorics*, 29(4):1012–1024, 2008.

28 Jaroslav Nešetřil and Patrice Ossona de Mendez. First order properties on nowhere dense structures. *The Journal of Symbolic Logic*, 75(03):868–887, 2010.

29 Jaroslav Nešetřil and Patrice Ossona de Mendez. On nowhere dense graphs. *European Journal of Combinatorics*, 32(4):600–617, 2011.

30 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity — Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.

31 Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. *ACM Trans. Algorithms*, 9(1):11, 2012.

32 Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. On wideness and stability. *arXiv preprint arXiv:1705.09336*, 2017.

33 Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972.

34 Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972.

The Intersection Problem for Finite Monoids

Lukas Fleischer
FMI, University of Stuttgart,
Universitätsstraße 38, 70569 Stuttgart, Germany
fleischer@fmi.uni-stuttgart.de

Manfred Kufleitner
Department of Computer Science, Loughborough University,
Epinal Way, Loughborough LE11 3TU, United Kingdom
m.kufleitner@lboro.ac.uk

Abstract

We investigate the intersection problem for finite monoids, which asks for a given set of regular languages, represented by recognizing morphisms to finite monoids from a variety $\mathbf{V}$, whether there exists a word contained in their intersection. Our main result is that the problem is PSPACE-complete if $\mathbf{V} \not\subseteq \mathbf{DS}$ and NP-complete if $\mathbf{1} \subsetneq \mathbf{V} \subseteq \mathbf{DO}$. Our NP-algorithm for the case $\mathbf{V} \subseteq \mathbf{DO}$ uses novel methods, based on compression techniques and combinatorial properties of $\mathbf{DO}$. We also show that the problem is log-space reducible to the intersection problem for deterministic finite automata (DFA) and that a variant of the problem is log-space reducible to the membership problem for transformation monoids. In light of these reductions, our hardness results can be seen as a generalization of both a classical result by Kozen and a theorem by Beaudry, McKenzie and Thérien.

2012 ACM Subject Classification Theory of computation → Regular languages, Theory of computation → Algebraic language theory, Theory of computation → Problems, reductions and completeness

Keywords and phrases Intersection Problem, Finite Monoid, Recognizing Morphism, Complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.30

Funding This work was supported by the DFG grants DI 435/5-2 and KU 2716/1-1.

1 Introduction

In 1977, Kozen showed that deciding whether the intersection of the languages recognized by a set of given deterministic finite automata (DFA) is non-empty is PSPACE-complete [16]. This result has since been the building block for numerous hardness results in formal language theory and related fields; see e.g. [8, 11, 12, 15]. It is natural to ask whether the problem becomes easier when restricting the input. Various special cases, such as bounding the number k of automata in the input [17] or considering only automata with a fixed number of accepting states [9], were investigated in follow-up work; see [14] for a survey.

Another very natural restriction is to only consider automata with certain *structural properties*. One such property is *counter-freeness*: an automaton is *counter-free* if no word permutes a non-trivial subset of its states. By a famous result of Schützenberger [19], a regular language is recognized by a counter-free automaton if and only if it is star-free. These properties are often expressed using the algebraic framework: instead of considering the automaton itself, one considers its transition monoid. The latter is the transformation monoid generated by the action of the letters on the set of states. Now, properties of automata

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 30; pp. 30:1–30:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


Table 1 Summary of complexity results (new main result, follows from reductions.)

	MonIsect(**V**)	MonIsect$_1$(**V**)	Memb(**V**) [2, 7]
NC	—	$\mathbf{V} \subseteq \mathbf{G}$	$\mathbf{V} \subseteq \mathbf{G}$
P	—	$\mathbf{V} \subseteq \mathbf{R_1} \vee \mathbf{L_1}$	$\mathbf{V} \subseteq \mathbf{R_1} \vee \mathbf{L_1}$
NP	$\mathbf{V} \subseteq \mathbf{DO}$	$\mathbf{V} \subseteq \mathbf{DO}$	$\mathbf{V} \subseteq \mathbf{R}, \mathbf{V} \subseteq \mathbf{A_1}$
NP-hard	all $\mathbf{V} \neq \mathbf{1}$	—	$\mathbf{Acom_2} \subseteq \mathbf{V}$, $\mathbf{XR} \subseteq \mathbf{V}, \mathbf{XL} \subseteq \mathbf{V}$
PSPACE	all **V**	all **V**	all **V**
PSPACE-hard	$\mathbf{V} \not\subseteq \mathbf{DS}$	$\mathbf{V} \not\subseteq \mathbf{DS}$	$\mathbf{V} \not\subseteq \mathbf{DS}$

can be given by membership of the transition monoid in certain classes, so-called *varieties*, of finite monoids. For example, an automaton is counter-free if and only if its transition monoids belongs to the variety **A** of *aperiodic* monoids. The DFA intersection problem for a variety **V**, denoted by DfaIsect(**V**), is formalized as follows.

DfaIsect(**V**)	
Input:	DFAs $A_1, \ldots, A_k$ with transition monoids from **V**
Question:	Is $L(A_1) \cap \cdots \cap L(A_k) \neq \emptyset$?

Note that DfaIsect(**Mon**), where **Mon** is the variety of all finite monoids, is the general DFA intersection problem considered by Kozen. A careful inspection of his proof actually reveals that DfaIsect(**A**) is PSPACE-complete already [11]. Additionally requiring all DFAs to have a single accepting state, we obtain a variant of DfaIsect(**V**) reminiscent of another problem investigated by Kozen, the *membership problem for transformation monoids*.

Memb(**V**)	
Input:	Transformations $f_1, \ldots, f_m \colon X \to X$ generating a monoid $T \in \mathbf{V}$ and $g \colon X \to X$
Question:	Does g belong to T?

The complexity of Memb(**V**) was studied extensively in a series of papers [2, 4, 5, 6, 7, 13, 3]. However, for certain varieties **V**, obtaining the exact complexity of DfaIsect(**V**) and Memb(**V**) is a challenging problem. To date, only partial results are known, see Table 1. For example, it is open whether or not Memb(**DA**) $\in$ NP, a question stated explicitly in [7] and revisited in [20] around ten years later.

Since algebraic tools are already used to express structural properties of automata, it seems natural to consider the fully algebraic version of the intersection problem by directly using finite monoids as language acceptors instead of taking the detour via automata and their transition monoids. A language $L \subseteq A^*$ is *recognized* by a morphism $h \colon A^* \to M$ to a finite monoid M if $L = h^{-1}(P)$ for some subset P of M. The set P is often called the *accepting set* because it resembles the accepting states in finite automata. A monoid M recognizes a language $L \subseteq A^*$ if there exists a morphism $h \colon A^* \to M$ recognizing L. It is well-known that a language is recognized by a finite monoid if and only if it is regular. For a variety of finite monoids **V**, the *intersection problem for* **V** is defined as follows.

MonIsect(**V**)	
Input:	Morphisms $h_i \colon A^* \to M_i \in \mathbf{V}$ and sets $P_i \subseteq M_i$ with $1 \leqslant i \leqslant k$
Question:	Is $h_1^{-1}(P_1) \cap \cdots \cap h_k^{-1}(P_k) \neq \emptyset$?

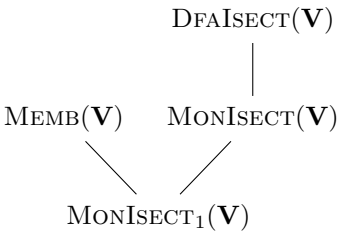


Figure 1 Relations between the problems considered in this work.

We assume that the monoids are given as multiplication tables, such that, assuming a random-access machine model, multiplications can be performed in logarithmic time.

There is a close connection to both the DFA intersection problem and the membership problem for transformation monoids. More specifically, for every variety $\mathbf{V}$, there is a log-space reduction of $\textsc{MonIsect}(\mathbf{V})$ to $\textsc{DfaIsect}(\mathbf{V})$. The variant $\textsc{MonIsect}_1(\mathbf{V})$ of the finite monoid intersection problem, where each of the accepting sets is a singleton, can be reduced to $\textsc{Memb}(\mathbf{V})$. Our reducibility results are depicted in Figure 1.

Not only is the algebraic version of the intersection problem a natural problem to consider, making progress in classifying its complexity also raises hope to make progress in solving open complexity questions regarding $\textsc{DfaIsect}(\mathbf{V})$ and $\textsc{Memb}(\mathbf{V})$. Using novel techniques, we prove that $\textsc{MonIsect}(\mathbf{V})$ is NP-complete whenever $\mathbf{V} \subseteq \mathbf{DO}$ and PSPACE-complete whenever $\mathbf{V} \not\subseteq \mathbf{DS}$. In particular, since $\mathbf{DA}$ is a subset of $\mathbf{DO}$, we obtain an NP-algorithm for $\textsc{MonIsect}(\mathbf{DA})$ while the problem of whether there exists such an algorithm for $\textsc{Memb}(\mathbf{DA})$ or $\textsc{DfaIsect}(\mathbf{DA})$ has been open for more than 25 years. Moreover, in view of the reductions mentioned above, our PSPACE-hardness result can be seen as a generalization of both Kozen's result and a result from [7], stating that every variety of aperiodic monoids not contained within $\mathbf{DA} = \mathbf{DS} \cap \mathbf{A}$ admits a PSPACE-complete transformation monoid membership problem.

Our results are summarized in Table 1. Only a very small gap of varieties contained within $\mathbf{DS}$ but not $\mathbf{DO}$ remains. Answering complexity questions in this setting is deeply connected to understanding the languages recognized by monoids in $\mathbf{DS}$ which is another problem open for over twenty years; see e.g. [1, Open Problem 14]. Obtaining a dichotomy result for $\textsc{MonIsect}(\mathbf{V})$ is likely to provide new major insights for both $\textsc{DfaIsect}(\mathbf{V})$ and the language variety corresponding to $\mathbf{DS}$, and, conversely, new insights on either language properties of $\mathbf{DS}$ or on $\textsc{DfaIsect}(\mathbf{DS})$ will potentially help with obtaining such a result.

We conclude with a first complexity result on the intersection problem for finite monoids.

▶ **Theorem 1.** $\textsc{MonIsect}(\mathbf{Mon}) \in$ PSPACE.

Proof. Since PSPACE = NPSPACE by Savitch's Theorem, it suffices to give a non-deterministic algorithm which requires polynomial space. The algorithm proceeds by guessing a word in the intersection, letter by letter. The word is not written down explicitly but after each guess, the image of the current prefix under each morphism is computed and stored. Finally, the algorithm verifies that each of the images is in the corresponding accepting set. ◀

2 Preliminaries

Words and Languages. Let A be a finite alphabet. A *word* over A is a finite sequence of letters $a_1 \cdots a_\ell$ with $a_i \in A$ for all $i \in \{1, \ldots, \ell\}$. The set A^* denotes the set of all words over A and a *language* is a subset of A^*. The *content* (or *alphabet*) of a word $w = a_1 \cdots a_\ell$

is the subset $\mathsf{alph}(w) = \{a_1, \ldots, a_\ell\}$ of A. A word u is a *factor* of w if there exist $p, q \in A^*$ such that $w = puq$; and, when the factorization is fixed, then the position of u is called its *occurrence*.

Algebra. Let M be a finite monoid. An element $e \in M$ is *idempotent* if $e^2 = e$. The set of all idempotent elements of M is denoted by $E(M)$. In a finite monoid M, the integer $\omega_M = |M|!$ plays an important role: for each $m \in M$, the element m^{ω_M} is idempotent. For convenience, we often write ω instead of ω_M if the reference to M is clear from the context. For two elements $m, n \in M$, we write $m \leqslant_{\mathcal{J}} n$ if the two-sided ideal of m is contained in the two-sided ideal of n, i.e., $MmM \subseteq MnM$. We write $m \mathcal{J} n$ if both $m \leqslant_{\mathcal{J}} n$ and $n \leqslant_{\mathcal{J}} m$.

The *direct product* of two monoids M and N is the Cartesian product $M \times N$ with componentwise multiplication. A monoid N is a *quotient* of a monoid M if there exists a surjective morphism $h\colon M \to N$. A monoid N is a *divisor* of a monoid M if N is a quotient of a submonoid of M.

A *variety* of finite monoids is a class $\mathbf{V}$ of finite monoids which is closed under (finite) direct products and divisors. The class of all finite monoids **Mon** is a variety. The following other varieties play an important role in this work:

$$\mathbf{G} = \{M \in \mathbf{Mon} \mid \forall e \in E(M) : e = 1\}$$
$$\mathbf{DS} = \{M \in \mathbf{Mon} \mid \forall e, f \in E(M) : e \mathcal{J} f \implies (efe)^\omega = e\}$$
$$\mathbf{DO} = \{M \in \mathbf{Mon} \mid \forall e, f \in E(M) : e \mathcal{J} f \implies efe = e\}$$

It is easy to see that $\mathbf{G}$ contains exactly those finite monoids which are groups. Since direct products of groups are groups and divisors of groups are groups, $\mathbf{G}$ is indeed a variety. For proofs that **DS** and **DO** are varieties, we refer to [18]. From the definitions, it follows immediately that $\mathbf{DO} \subseteq \mathbf{DS}$. There exist several other interesting characterizations of **DS**. Let B_2^1 be the monoid defined on the set $\{1, a, b, ab, ba, 0\}$ by the operation $aba = a$, $bab = b$ and $a^2 = b^2 = 0$ where 0 is a zero element. Then the following holds, see e.g. [1].

▶ **Proposition 2.** *Let M be a finite monoid. The following properties are equivalent:*
1. *$M \in \mathbf{DS}$.*
2. *For each $e \in E(M)$ and $x \in M$ with $e \leqslant_{\mathcal{J}} x$, we have $(exe)^\omega = e$.*
3. *For each $e \in E(M)$, the elements $\{x \in M \mid e \leqslant_{\mathcal{J}} x\}$ form a submonoid of M.*
4. *B_2^1 is not a divisor of $M \times M$.*

Tiling Systems. A *tiling system* is a tuple $\mathcal{T} = (\Lambda, T, n, f, b)$ where Λ is a finite set of *labels*, $T \subseteq \Lambda \times \Lambda \times \Lambda \times \Lambda$ are the so-called *tiles*, $n \in \mathbb{N}$ is the *width* and $f, b \in T^n$ are the *first row* and *bottom row*. For a tile $t = (t_w, t_e, t_s, t_n) \in T$, we let $\lambda_w(t) = t_w$, $\lambda_e(t) = t_e$, $\lambda_s(t) = t_s$ and $\lambda_n(t) = t_n$. These labels can be thought of as labels in *west*, *east*, *south* and *north* direction. An *m-tiling* of $\mathcal{T}$ is a mapping $\tau\colon \{1, \ldots, m\} \times \{1, \ldots, n\} \to T$ such that the following properties hold:
1. $\tau(1,1)\tau(1,2)\cdots\tau(1,n) = f$,
2. $\lambda_e(\tau(i,j)) = \lambda_w(\tau(i,j+1))$ for $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n-1$,
3. $\lambda_s(\tau(i,j)) = \lambda_n(\tau(i+1,j))$ for $1 \leqslant i \leqslant m-1$ and $1 \leqslant j \leqslant n$,
4. $\tau(m,1)\tau(m,2)\cdots\tau(m,n) = b$.

The *corridor tiling problem* asks for a given tiling system $\mathcal{T}$ whether there exists some $m \in \mathbb{N}$ such that there is a m-tiling of $\mathcal{T}$. The *square tiling problem* asks for a given tiling system $\mathcal{T}$ of width n, whether there exists an n-tiling of $\mathcal{T}$. It is well-known that the corridor tiling problem is PSPACE-complete and that the square tiling problem is NP-complete [10].

Straight-Line Programs. A *straight-line program* (*SLP*) is a grammar $S = (V, A, P, X_s)$ where V is a finite set of variables, A is a finite alphabet, $P\colon V \to (V \cup A)^*$ is a mapping and $X_s \in V$ is the so-called *start variable*. For a variable $X \in V$, the word $P(X)$ is the *right-hand side* of X. We require that there exists a linear order $<$ on V such that $Y < X$ whenever $P(X) \in (V \cup A)^* Y (V \cup A)^*$. Starting with some word $\alpha \in (V \cup A)^*$ and repeatedly replacing variables $X \in V$ by $P(X)$ yields a word from A^*, the so called *evaluation of* α, denoted by $\mathsf{val}(\alpha)$. The word *produced by* S is $\mathsf{val}(S) = \mathsf{val}(X_s)$. If the reference to A and V is clear from the context, we will often use the notation $h(\alpha)$ instead of $h(\mathsf{val}(\alpha))$ for the image of the evaluation of a word $\alpha \in (A \cup V)^*$ under a morphism $h\colon A^* \to M$. Analogously, we write $h(S)$ instead of $h(\mathsf{val}(S))$. The *size* of S is $|S| = \sum_{X \in V} |P(X)|$. Each variable X of an SLP S can be viewed as an SLP itself by making X the start variable of S.

The following simple lemma illustrates how SLPs can be used for compression.

▶ **Lemma 3.** *Let $S = (V, A, P, X_s)$ be an SLP and let $e \in \mathbb{N}$. Let w be the word produced by S. Then there exists an SLP S' of size $|S'| \leqslant |S| + 4\log(e)$ such that S' produces w^e.*

Proof. We obtain S' by iteratively adding new variables to V as follows, starting with $i = e$ and repeating the process until $i = 0$.

- If $i > 0$ is odd, add a new variable X_i and let $P(X_i) = X_{i-1}X_s$. Let $i := i - 1$.
- If $i > 0$ is even, add a new variable X_i and let $P(X_i) = X_{i/2}X_{i/2}$. Let $i := i/2$.

Finally, add the variable X_0 and let $P(X_0) = \varepsilon$. The new start variable is X_e and by construction, we have $\mathsf{val}(X_e) = w^e$. ◀

3 Connections to Other Problems

Before investigating the complexity of $\textsc{MonIsect}(\mathbf{V})$ itself, we establish connections to other well-known problems defined in the introduction, starting with the DFA intersection problem.

▶ **Proposition 4.** *Let $\mathbf{V}$ be a variety of finite monoids, let $M \in \mathbf{V}$, let $h\colon A^* \to M$ be a morphism and let $P \subseteq M$. Then there exists a finite deterministic automaton A with $|M|$ states such that $L(A) = h^{-1}(P)$ and such that the transition monoid of A belongs to $\mathbf{V}$. When the monoid, the morphism and the accepting set are given as inputs, this automaton is log-space computable.*

Proof. It suffices to perform the standard conversion of monoids to finite automata. The set of states of A is M, the initial state is the identity element 1, the transitions are defined by $\delta(m, a) = mh(a)$ for all $m \in M$ and $a \in A$ and the accepting states are P. A straightforward verification shows that the transition monoid of A is isomorphic to M. Since computing images $h(a)$ and performing multiplications are just table lookups, each output bit can be computed in logarithmic time on a random-access machine model. ◀

▶ **Corollary 5.** *For each variety of finite monoids $\mathbf{V}$, the problem $\textsc{MonIsect}(\mathbf{V})$ is log-space reducible to $\textsc{DfaIsect}(\mathbf{V})$.*

For a direct link to $\textsc{Memb}(\mathbf{V})$, we consider the variant $\textsc{MonIsect}_1(\mathbf{V})$ of the finite monoid intersection problem. In this variant, each of the accepting sets is a singleton.

▶ **Proposition 6.** *Let $\mathbf{V}$ be a variety of finite monoids and let $M_1, \ldots, M_k \in \mathbf{V}$ be pairwise disjoint finite monoids. For each $i \in \{1, \ldots, k\}$, let $h_i : A^* \to M_i$ be a morphism and let $p_i \in M_i$. Then there exists a transformation monoid $T \in \mathbf{V}$ on the set $M = M_1 \cup \cdots \cup M_k$, a morphism $h\colon A^* \to T$ and a transformation $p \in T$ such that $h^{-1}(p) = h_1^{-1}(p_1) \cap \cdots \cap h_k^{-1}(p_k)$.*

Proof. For each $a \in A$, we define a transformation $f_a : M \to M$ by $f_a(m) = mh_i(a)$ for $m \in M_i$. The closure of $\{f_a \mid a \in A\}$ under composition is the transformation monoid T and the morphism $h\colon A^* \to T$ is given by $h(a) = f_a$. We let $p\colon M \to M$ be the transformation defined by $p(m) = mp_i$ for $m \in M_i$.

We need to verify that $h^{-1}(p) = h_1^{-1}(p_1) \cap \cdots \cap h_k^{-1}(p_k)$. For the inclusion from right to left, let $w \in A^*$ be a word such that $h_i(w) = p_i$ for each $i \in \{1, \ldots, k\}$. Then, by definition, $h(w)$ is the transformation which maps an element $m \in M_i$ to $mh_i(w) = mp_i$, i.e., $h(w) = p$. The converse inclusion is trivial.

It is easy to check that T is a divisor of the direct product $M_1 \times \cdots \times M_k$ and thus, by closure of $\mathbf{V}$ under direct products and under division, T belongs to $\mathbf{V}$ as well. Since computing images $h_i(a)$ and performing multiplications are just table lookups, each output bit can be computed in logarithmic time on a random-access machine model. ◀

▶ **Corollary 7.** *For each variety of finite monoids* $\mathbf{V}$*, the problem* $\textsc{MonIsect}_1(\mathbf{V})$ *is log-space reducible to* $\textsc{Memb}(\mathbf{V})$*.*

4 Hardness Results

The following lower bound can be viewed as a variant of classical NP-hardness results and is based on the well-known fact that each non-trivial variety contains either the monoid $U_1 = \{0, 1\}$ with integer multiplication or a finite cyclic group (however, the proof itself does not require this case distinction).

▶ **Theorem 8.** *Let* $\mathbf{V}$ *be a non-trivial variety of finite monoids. Then, the decision problem* $\textsc{MonIsect}(\mathbf{V})$ *is* NP*-hard.*

Proof. We give a polynomial-time reduction of the square tiling problem to $\textsc{MonIsect}(\mathbf{V})$.

Let $\mathcal{T} = (\Lambda, T, n, f, b)$ be a tiling system. Let $M \in \mathbf{V}$ be a non-trivial finite monoid and let $x \in M \setminus \{1\}$. The alphabet A is the set $T \times \{1, \ldots, n\} \times \{1, \ldots, n\}$. Let $f = t_1 \cdots t_n$. For each integer $j \in \{1, \ldots, n\}$ and each direction $d \in \{w, e, s, n\}$, we define a morphism $f_{j,d}\colon A \to M$ by mapping $(t, 1, j)$ to x if $\lambda_d(t) = \lambda_d(t_j)$ and mapping the remaining letters to 1. Analogously, with $b = u_1 \cdots u_n$, we let $b_{j,d}\colon A \to M$ be the morphism mapping (t, n, j) to x if $\lambda_d(t) = \lambda_d(u_j)$ and mapping other letters to 1. For each integer $i \in \{1, \ldots, n\}$, each $j \in \{1, \ldots, n-1\}$ and each label $\mu \in \Lambda$, we define a morphism $h_{i,j,\mu}\colon A \to M \times M$ by

$$h_{i,j,\mu}(t,k,\ell) = \begin{cases} (x,1) & \text{if } k = i,\ \ell = j \text{ and } \lambda_e(t) = \mu \\ (1,x) & \text{if } k = i,\ \ell = j+1 \text{ and } \lambda_w(t) = \mu \\ (1,1) & \text{otherwise} \end{cases}$$

and, analogously, we define morphisms $v_{i,j,\mu}\colon A \to M \times M$ with $i \in \{1, \ldots, n-1\}$ and $j \in \{1, \ldots, n\}$ and $\mu \in \Lambda$ as follows:

$$v_{i,j,\mu}(t,k,\ell) = \begin{cases} (x,1) & \text{if } k = i,\ \ell = j \text{ and } \lambda_s(t) = \mu \\ (1,x) & \text{if } k = i+1,\ \ell = j \text{ and } \lambda_n(t) = \mu \\ (1,1) & \text{otherwise} \end{cases}$$

Finally, we define morphisms $g_{i,j,d,\mu,\mu'}\colon A \to M \times M$ with $i, j \in \{1, \ldots, n\}$, $d \in \{w, e, s, n\}$ as well as $\mu, \mu' \in \Lambda$ and $\mu \neq \mu'$ as follows:

$$g_{i,j,d,\mu,\mu'}(t,k,\ell) = \begin{cases} (x,1) & \text{if } k = i,\ \ell = j \text{ and } \lambda_d(t) = \mu \\ (1,x) & \text{if } k = i,\ \ell = j \text{ and } \lambda_d(t) = \mu' \\ (1,1) & \text{otherwise} \end{cases}$$

For each of the morphisms $b_{j,d}$ and $f_{j,d}$, the accepting set is $\{x\}$. For each $h_{i,j,\mu}$ and $v_{i,j,\mu}$, the accepting set is $\{(1,1),(x,x)\}$. The accepting set for each $g_{i,j,d,\mu,\mu'}$ is $\{(1,1),(1,x),(x,1)\}$. ◀

The next objective is to obtain a stronger result in the case that $\mathbf{V}$ contains some finite monoid which is not in $\mathbf{DS}$. Our proof is based on the well-known fact that direct products of B_2^1 can be used to encode computations of a Turing machine or runs of an automaton, an idea which already appears in the proof of [7, Theorem 4.9]. To this end, we first describe classes of languages recognizable by such direct products.

▶ **Lemma 9.** *Let $\mathbf{V}$ be a variety of finite monoids such that $\mathbf{V} \not\subseteq \mathbf{DS}$. Let A be a finite alphabet and let B, C, D, E, F be (possibly empty) pairwise disjoint subsets of A. Then, each of the languages $E^*B(D \cup E)^*$, $(D \cup E)^*CE^*$ and $(E^*B(E \cup F)^*CE^* \cup E^*DE^*)^+$ is the preimage of an element of a monoid $M \in \mathbf{V}$ of size 6 under a morphism $h\colon A^* \to M$.*

Proof. Let N be a monoid from $\mathbf{V} \setminus \mathbf{DS}$. By Proposition 2, the monoid B_2^1 is a divisor of the direct product $N \times N$ and since $\mathbf{V}$ is closed under direct products and divisors, we have $B_2^1 \in \mathbf{V}$. We let $M = B_2^1$.

For $E^*B(D \cup E)^*$, consider the morphism $h\colon A^* \to M$ defined by $h(e) = 1$ for $e \in E$, $h(b) = b$ for $b \in B$, $h(d) = ab$ for all $d \in D$. All other letters are mapped to the zero element. By construction, we have $h^{-1}(b) = E^*B(D \cup E)^*$. For $(D \cup E)^*CE^*$, one can use a symmetrical construction.

For $(E^*B(E \cup F)^*CE^* \cup E^*DE^*)^+$, we define $h\colon A^* \to M$ by $h(b) = a$ for all $b \in B$, $h(c) = b$ for $c \in C$, $h(d) = ab$ for $d \in D$, $h(f) = ba$ for $f \in F$ and $h(e) = 1$ for $e \in E$. Again, the remaining letters are mapped to 0. The preimage of ab is the desired language. ◀

▶ **Lemma 10.** *Let $\mathbf{V}$ be a variety of finite monoids such that $\mathbf{V} \not\subseteq \mathbf{DS}$. Let A be a finite alphabet, let $n \in \mathbb{N}$ and let $A_1, \ldots, A_n$ be pairwise disjoint subsets of A. Then the language $(A_1 \cdots A_n)^+$ can be written as an intersection of n languages, each of which is the preimage of an element of a monoid $M \in \mathbf{V}$ of size 6 under a morphism $h\colon A^* \to M$.*

Proof. Let $B = A_1 \cup \cdots \cup A_n$. For $1 \leqslant i \leqslant n-1$, we define the alphabet $D_i = B \setminus (A_i \cup A_{i+1})$ and the language $L_i = (A_iA_{i+1} \cup D_i)^+$. We also let $L_n = (A_1D_n^*A_n)^+$ with $D_n = B \setminus (A_1 \cup A_n)$. By construction, we have $L_1 \cap \cdots \cap L_n = (A_1 \cdots A_n)^+$ and by Lemma 9, each of the languages L_i is recognized by a monoid of size 6. ◀

We are now able to state the second main theorem of this section.

▶ **Theorem 11.** *Let $\mathbf{V}$ be a variety of finite monoids such that $\mathbf{V} \not\subseteq \mathbf{DS}$. Then, the decision problem* $\textsc{MonIsect}_1(\mathbf{V})$ *is* PSPACE*-complete.*

Proof. Let $\mathcal{T} = (\Lambda, T, n, f, b)$ be a tiling system. The objective is to construct a language L which is non-empty if and only if there exists a valid m-tiling of $\mathcal{T}$ for some $m \in \mathbb{N}$.

We may assume without loss of generality that $\lambda_w(t) \neq \lambda_e(t)$ and $\lambda_s(t) \neq \lambda_n(t)$ for all tiles $t \in T$. If, for example, $\lambda_w(t) = \mu = \lambda_e(t)$ for a tile $t \in T$, we create a copy μ' of the label μ and replace every tile with $\lambda_w(t) = \mu$ by two copies. In one of the copies, we replace the west label with μ'. We repeat this for all other directions and finally remove all tiles with $\lambda_w(t) = \lambda_e(t) \in \{\mu, \mu'\}$.

We define an alphabet $A = T \times \{0,1,2\} \times \{1, \ldots, n\}$. Intuitively, the letters of A correspond to positions in a tiling. The first component describes the tile itself, the second component specifies whether the tile is in the first row, some intermediate row or in the

bottom row and the third component specifies the column. For each $j \in \{1, \ldots, n\}$ and $\mu \in \Lambda$, let $C_j = T \times \{0,1,2\} \times \{j\}$ and $D_j = A \setminus C_j$ and

$$W_\mu = \{(t,i,j) \in A \mid \lambda_w(t) = \mu, j > 1\}, \qquad N_{j,\mu} = \{(t,i,j) \in A \mid \lambda_n(t) = \mu, i > 0\},$$
$$E_\mu = \{(t,i,j) \in A \mid \lambda_e(t) = \mu, j < n\}, \qquad S_{j,\mu} = \{(t,i,j) \in A \mid \lambda_s(t) = \mu, i < 2\},$$
$$X_\mu = A \setminus (W_\mu \cup E_\mu), \qquad Y_{j,\mu} = C_j \setminus (N_{j,\mu} \cup S_{j,\mu}).$$

Note that by our initial assumption, $W_\mu \cap E_\mu = \emptyset$ and $N_{j,\mu} \cap S_{j,\mu} = \emptyset$ for each $\mu \in \Lambda$ and for $1 \leqslant j \leqslant n$. Let $F_j = \{(t_j, 0, j)\}$ and $B_j = \{(u_j, 2, j)\}$ where t_j and u_j are the tiles uniquely determined by $f = t_1 \cdots t_n$ and $b = u_1 \cdots u_n$. Let $\overline{F}_j = \{(t,i,j) \in A \mid i > 0\}$ and $\overline{B}_j = \{(t,i,j) \in A \mid i < 2\}$. We define

$$K = \left(\bigcap_{1 \leqslant j \leqslant n} D_j^* F_j (\overline{F}_j \cup D_j)^* \right) \cap \left(\bigcap_{1 \leqslant j \leqslant n} (\overline{B}_j \cup D_j)^* B_j D_j^* \right) \cap \left(\bigcap_{\mu \in \Lambda} (E_\mu W_\mu \cup X_\mu)^+ \right)$$
$$\cap \left(\bigcap_{\substack{\mu \in \Lambda, \\ 1 \leqslant j \leqslant n}} (D_j^* S_{j,\mu} D_j^* N_{j,\mu} D_j^* \cup D_j^* Y_{j,\mu} D_j^*)^+ \right).$$

and $L = (C_1 \cdots C_n)^+ \cap K$. By Lemma 9 and Lemma 10, the language L can be represented by a MonIsect($\mathbf{V}$) instance with polynomially many morphisms to monoids of size 6 from $\mathbf{V}$ and with singleton accepting sets. ◀

5 A Small Model Property for DO

The objective of this section is to prove the following result which states that, within a non-empty intersection of languages recognized by monoids from **DO**, there always exists a word with a small SLP representation.

▶ **Theorem 12.** *For each $i \in \{1, \ldots, k\}$, let $M_i \in \mathbf{DO}$ and let $h_i \colon A^* \to M_i$ be a morphism. Let $w \in A^*$. Then there exists an SLP S of size at most $p(N)$ with $h_i(S) = h_i(w)$ for all $i \in \{1, \ldots, k\}$ where $p \colon \mathbb{R} \to \mathbb{R}$ is some polynomial and $N = |M_1| + \cdots + |M_k|$.*

Before diving into the proof of this result, we note that the theorem immediately yields the following corollary:

▶ **Corollary 13.** MonIsect(**DO**) *is* NP*-complete.*

Proof. In view of Theorem 8, it suffices to describe an NP-algorithm. The algorithm first non-deterministically guesses an SLP of polynomial size producing a word in the intersection of the given languages. It remains to check that the word represented in the SLP is indeed contained in each of the languages. To this end, we compute the image of the word represented by the SLP under each of the morphisms. Each such computation can be performed in time linear in the size of the SLP by computing the image of a variable X as soon as the images of all variables appearing on the right-hand side of X are computed already, starting with minimal variables. ◀

5.1 The Group Case

We first take care of a special case, namely that each of the monoids is a group. In this case, one can use a variant of the *Schreier-Sims algorithm* [3, 13] to obtain a compressed

representative. To keep the paper self-contained, we give the full algorithm alongside with a correctness proof.

Our setting is as follows: the input are groups $G_1, \ldots, G_k$ which are, without loss of generality, assumed to be pairwise disjoint, and morphisms $h_i \colon A^* \to G_i$ with $i \in \{1, \ldots, k\}$. We let $G = G_1 \cup \cdots \cup G_k$ and $N = |G|$. Note that G is considered as a set; it does not form a group unless $k = 1$. However, for each $g \in G$, we interpret powers g^i in the corresponding group G_i with $g \in G_i$. We let $\omega = N!$ so that, for each $g \in G$, the element g^ω is the identity.[1]

Algorithm 1 The SIFT procedure.

```
procedure SIFT(α)
    R_0 ← ε
    for i ∈ {1, ..., k} do
        S_i ← R_{i-1}^{ω-1} α
        if T[h_i(S_i)] = ε then T[h_i(S_i)] ← S_i end if
        R_i ← R_{i-1} T[h_i(S_i)]
    end for
    return R_k
end procedure
```

The algorithm maintains a table $T \colon G \to (A \cup V)^*$ as an internal data structure, where the set of variables V is extended as needed and the table entries $T[g]$ can be considered variables themselves. The SIFT procedure expects a parameter $\alpha \in (V \cup A)^*$ and tries to find a short representation of $\mathsf{val}(\alpha)$, using only entries from the table unless it comes across an empty table entry, in which case it uses α to fill the missing table entry itself. When a table entry is assigned a word with a factor of the form $X^{\omega-1}$, this factor is stored in a compressed form by using the technique from Lemma 3 and adding new variables as needed. Thus, a factor $X^{\omega-1}$ only requires $4\log(\omega - 1) \leqslant 4\log(N!) \leqslant 4N\log(N)$ additional space.

Algorithm 2 Initialization of the compression algorithm for groups.

```
procedure INIT
    for all g ∈ G do T[g] ← ε end for
    c ← 0
    repeat
        c_p ← c
        for all g_1 ∈ G_1, ... g_k ∈ G_k, a ∈ A do
            SIFT(T[g_1] ··· T[g_k] a)
        end for
        c ← |{g ∈ G | T[g] ≠ ε}|
    until c = c_p
end procedure
```

Before the SIFT procedure is used for compression, the table needs to be initialized. To this end, the INIT routine fills the table with short representatives such that future SIFT invocations never run into empty table entries again. Let us first prove several invariants of the SIFT procedure.

[1] One could also choose $\omega = \operatorname{lcm}\{|G_1|, \ldots, |G_k|\}$ but for the analysis, it does not matter, since $N!$ is sufficiently small.

▶ **Lemma 14.** *For each $i \in \{1, \ldots, k\}$ and $g \in G_i$, we have $T[g] = \varepsilon$ or $h_i(T[g]) = g$.*

Proof. Suppose that $T[g] \neq \varepsilon$. Then, in some round of the SIFT procedure, we have $h_i(S_i) = g$ and $T[h_i(S_i)]$ is assigned the SLP S_i (and never modified again). Therefore, $h_i(T[g]) = h_i(T[h_i(S_i)]) = h_i(S_i) = g$. ◀

▶ **Lemma 15.** *After round i of the* SIFT *procedure, we have $h_i(R_i) = h_i(\alpha)$.*

Proof. By the definition of R_i, we have $h_i(R_i) = h_i(R_{i-1}T[h_i(S_i)])$ which is the same as $h_i(R_{i-1}S_i)$ by Lemma 14. Plugging in the definition of S_i yields $h_i(R_{i-1}R_{i-1}^{\omega-1}\alpha) = h_i(\alpha)$ where the latter equality holds since G_i is a group. ◀

▶ **Lemma 16.** *For $1 \leqslant i < j \leqslant k$ and for all $g \in G_j$, we have $h_i(T[g]) = 1$.*

Proof. Consider the invocation of the SIFT procedure where $T[g]$ is defined. In round j of this invocation, the entry $T[g]$ is assigned some SLP S_j with $h_j(S_j) = g$.

Therefore, $h_i(T[g]) = h_i(S_j) = h_i(R_{j-1}^{\omega-1}\alpha)$. Expanding R_{j-1} yields

$$h_i(R_{j-1}) = h_i\Big(\Big(\prod_{r=1}^{j-1} T[h_r(S_r)]\Big)\Big) = h_i\Big(\Big(\prod_{r=1}^{i} T[h_r(S_r)]\Big)\Big) = h_i(R_i)$$

where the second equality follows by induction. Therefore, $h_i(T[g]) = h_i(R_i^{\omega-1}\alpha)$ which is the same as $h_i(\alpha^{\omega-1}\alpha) = 1$ by Lemma 15. ◀

▶ **Lemma 17.** *After round j, we have $h_i(R_k) = h_i(\alpha)$ for all $i \in \{1, \ldots, j\}$.*

Proof. Using the expansion of R_k and Lemma 16, we obtain the sequence of equalities

$$h_i(R_k) = h_i\Big(\Big(\prod_{r=1}^{k} T[h_r(S_r)]\Big)\Big) = h_i\Big(\Big(\prod_{r=1}^{i} T[h_r(S_r)]\Big)\Big) = h_i(R_i).$$

The statement now follows immediately from Lemma 15. ◀

▶ **Theorem 18.** *For each $i \in \{1, \ldots, k\}$, let G_i be a finite group and let $h_i \colon A^* \to G_i$ be a morphism. Let $w \in A^*$. Then there exists an SLP S of size at most $p(N)$ with $h_i(S) = h_i(w)$ for all $i \in \{1, \ldots, k\}$ where $p \colon \mathbb{R} \to \mathbb{R}$ is some polynomial and $N = |G_1| + \cdots + |G_k|$.*

Proof. We claim that the SLP S constructed when calling INIT, followed by SIFT with parameter w satisfies the properties above. By Lemma 17, we have $h_i(S) = h_i(w)$ for all $i \in \{1, \ldots, k\}$. Moreover, when the initialization routine returns, the table entries contain SLP of polynomially bounded size. We now claim that any subsequent executions of the SIFT procedure will not define any new table entries, no matter which SLP is passed as a parameter. In particular, running SIFT(w) yields an SLP that only uses already existing table entries.

To prove the claim, assume, for the sake of contradiction, that there exists some word v such that some new table entry $T[g]$ is defined during SIFT(v). We choose v such that it is a word of minimal length satisfying this condition. This means that we can factorize $v = v'a$ with $a \in A$ such that all table entries are defined when calling SIFT(v'). Let $T[g_1] \cdots T[g_k]$ be the return value of SIFT(v'). Then SIFT$(T[g_1] \cdots T[g_k]a)$ is called during the initialization process and because $h_i(T[g_1] \cdots T[g_k]a) = h_i(v)$ for all $1 \leqslant i \leqslant k$, the sequence of S_i during the execution of SIFT$(T[g_1] \cdots T[g_k]a)$ is the same as in SIFT(v) which means that all table entries accessed during SIFT(v) are defined. ◀

5.2 The General Case

For the general case, where each of the monoids is in **DO** but not necessarily a group, we use combinatorial properties of languages recognized by monoids from **DO** to reduce the problem to the group case. The following lemmas are an essential ingredient of this reduction.

▶ **Lemma 19.** *Let $h\colon A^* \to M$ be a morphism to a finite monoid $M \in \mathbf{DS}$. Let $u, v \in A^*$ such that $h(v) \in E(M)$ and $\mathsf{alph}(u) \subseteq \mathsf{alph}(v)$. Then $h(v) \leqslant_{\mathcal{J}} h(u)$.*

Proof. Let $u = a_1 \cdots a_\ell$ with $a_i \in A$ for $1 \leqslant i \leqslant \ell$. Since $a_i \in \mathsf{alph}(v)$ for each $i \in \{1, \ldots, \ell\}$, we have $h(v) \leqslant_{\mathcal{J}} h(a_i)$. By Proposition 2, the set $\{x \in M \mid h(v) \leqslant_{\mathcal{J}} x\}$ is a submonoid of M which means that $h(v) \leqslant_{\mathcal{J}} h(a_1) \cdots h(a_\ell) = h(u)$, thereby proving the claim. ◀

▶ **Lemma 20.** *Let $M \in \mathbf{DO}$ and let $e, f, g \in E(M)$ with $e \mathcal{J} f \leqslant_{\mathcal{J}} g$. Then $egf = ef$.*

Proof. First note that $(fg)^\omega \mathcal{J} (fgf)^\omega \mathcal{J} (gf)^\omega$. Since $M \in \mathbf{DS}$, we have $(fgf)^\omega = f$ and since $M \in \mathbf{DO}$, we have $(fg)^\omega = (fg)^\omega (gf)^\omega (fg)^\omega = (fg)^{\omega-1}$. Together, this yields, $fgf = (fgf)^\omega gf = (fg)^\omega fgf = (fg)^{\omega-1} fgf = (fg)^\omega f = (fgf)^\omega = f$, thus $gf \in E(M)$. By Proposition 2, we obtain $gf \mathcal{J} e$. Therefore, $egf = eg(fef) = (egfe)f = ef$. ◀

▶ **Lemma 21.** *Let $M \in \mathbf{DO}$, let $e, f, g \in E(M)$ and let $x, y \in M$ such that $e \mathcal{J} f \leqslant_{\mathcal{J}} g, x, y$. Then $exgyf = exyf$.*

Proof. Since $M \in \mathbf{DS}$, we have $ex = (exe)^\omega x = ex(ex)^\omega$ and $yf = y(fyf)^\omega = (yf)^\omega yf$. Note that $(ex)^\omega \mathcal{J} e \mathcal{J} f \mathcal{J} (yf)^\omega$ by Proposition 2 and thus, Lemma 20 yields $(ex)^\omega g (yf)^\omega = (ex)^\omega (yf)^\omega$. Finally, combining all the equalities, we obtain the desired statement $exgyf = ex(ex)^\omega g (yf)^\omega yf = ex(ex)^\omega (yf)^\omega yf = exyf$. ◀

For the remainder of this section, let $M_1, \ldots, M_k \in \mathbf{DO}$ be finite monoids and let $h_i\colon A^* \to M_i$ be morphisms. We let $N = |M_1| + \cdots + |M_k|$. The occurrence of a word u in puq is called *isolated* if for each $i \in \{1, \ldots, k\}$, there exist words $v_i, w_i \in A^*$ such that

$$\mathsf{alph}(v_i) = \mathsf{alph}(w_i) \supseteq \mathsf{alph}(u), \quad h_i(pv_i) = h_i(p) \quad \text{and} \quad h_i(w_i q) = h_i(q).$$

Let $w = a_1 u_1 a_2 \cdots u_{\ell-1} a_\ell$ be a factorization of w with $a_j \in A$ and $u_j \in A^*$ for all $j \in \{1, \ldots, \ell\}$. Let $p_j = a_1 u_1 a_2 \cdots u_{j-1} a_j$ and $q_j = a_{j+1} u_{j+1} \cdots a_{\ell-1} u_{\ell-1} a_\ell$. The factorization $w = a_1 u_1 a_2 \cdots u_{\ell-1} a_\ell$ is called *piecewise isolating* if, for each $j \in \{1, \ldots, \ell-1\}$, the occurrence of u_j in $w = p_j u_j q_j$ is isolated. The value ℓ is the *length* of this factorization.

▶ **Lemma 22.** *Every word $w \in A^*$ admits a piecewise isolating factorization of length at most N^2.*

Proof. Let $w = b_1 \cdots b_m$ where $b_r \in A$ for $1 \leqslant r \leqslant m$. To each position $r \in \{1, \ldots, m\}$, we assign a set $C_r = \{(h_i(b_1 \cdots b_s), h_i(b_{s+1} \cdots b_m)) \mid 1 \leqslant i \leqslant k, 1 \leqslant s \leqslant r\}$. Note that by definition, we have $C_r \subseteq C_{r+1}$. Let $r_1, \ldots, r_\ell \in \mathbb{N}$ such that $r_1 = 1$, $r_\ell = m$ and $C_{r_{j-1}} = C_{r_j - 1} \subsetneq C_{r_j}$ for all $j \in \{2, \ldots, \ell\}$. Let $a_j = b_{r_j}$ and let $u_j = b_{r_j+1} \cdots b_{r_{j+1}-1}$ for all $j \in \{1, \ldots, \ell\}$.

Now, for $j \in \{1, \ldots, \ell\}$ and $i \in \{1, \ldots, k\}$, let $t(j,i)$ be the smallest index g such that $(h_i(a_1 u_1 \cdots a_j u_j), h_i(a_{j+1} u_{j+1} \cdots a_{\ell-1} u_{\ell-1} a_\ell)) \in C_{r_g}$, i.e., the prefix of length $r_{t(j,i)}$ of w is the shortest prefix p such that $w = pq$ for some $q \in A^*$ and the image of p under h_i is $h_i(a_1 u_1 \cdots a_j u_j)$ and the image of q is $h_i(a_{j+1} u_{j+1} \cdots a_{\ell-1} u_{\ell-1} a_\ell)$. Note that $t(j,i) \leqslant j$ and, by choice of $t(j,i)$, we have

$$h_i(a_1 u_1 \cdots a_j u_j) = h_i(a_1 u_1 a_2 \cdots u_{t(j,i)-1} a_{t(j,i)}) \text{ and} \tag{1}$$

$$h_i(a_{j+1} u_{j+1} \cdots a_{\ell-1} u_{\ell-1} a_\ell) = h_i(u_{t(j,i)} a_{t(j,i)+1} \cdots u_{\ell-1} a_\ell). \tag{2}$$

Let $w_{ji} = u_{t(j,i)}a_{t(j,i)+1}\cdots u_{j-1}a_ju_j$ and let $v_{ji} = u_ju_{t(j,i)}a_{t(j,i)+1}\cdots u_{j-1}a_j$. In the special case $t(j,i) = j$, we obtain $w_{ji} = v_{ji} = u_j$.

For $p_j = a_1u_1a_2\cdots u_{j-1}a_j$ and $q_j = a_{j+1}u_{j+1}\cdots a_{\ell-1}u_{\ell-1}a_\ell$, equation (1) implies

$$\begin{aligned} h_i(p_jv_{ji}) &= h_i(a_1u_1\cdots a_ju_j)\cdot h_i(u_{t(j,i)}a_{t(j,i)+1}\cdots u_{j-1}a_j) \\ &= h_i(a_1u_1a_2\cdots u_{t(j,i)-1}a_{t(j,i)})\cdot h_i(u_{t(j,i)}a_{t(j,i)+1}\cdots u_{j-1}a_j) = h_i(p_j) \end{aligned}$$

and, similarly, equation (2) yields $h_i(w_{ji}q_j) = h_i(q_j)$. Since u_j is a suffix of w_{ji} and since v_{ji} can be obtained by rotating w_{ji} cyclically, we have $\mathsf{alph}(v_{ji}) = \mathsf{alph}(w_{ji}) \supseteq \mathsf{alph}(u_j)$. The bound on ℓ follows from the fact that $C_{r_1} \subsetneq \cdots \subsetneq C_{r_\ell} \subseteq \bigcup_{i=1}^{k} M_i \times M_i$. ◀

The lemma above suggests that it is sufficient to construct SLPs for isolated occurrences. Thus, let now $u \in A^*$ be an isolated occurrence of $w = puq$, and let $B = \mathsf{alph}(u)$. For each $i \in \{1,\ldots,k\}$, we define an equivalence relation $\equiv_i$ on the submonoid $T_i = h_i(B^*)$ of M_i by $m \equiv_i n$ if and only if $h_i(p)xmyh_i(q) = h_i(p)xnyh_i(q)$ for all $x, y \in T_i$. It is easy to check that this relation is a congruence. Moreover, for all $u, v \in B^*$ with $h_i(u) \equiv_i h_i(v)$, we have $h_i(puq) = h_i(pvq)$. Another fundamental property of $\equiv_i$ is captured in the following lemma.

▶ **Lemma 23.** *For each $i \in \{1,\ldots,k\}$, the quotient $T_i/{\equiv_i}$ is a group.*

Proof. Let $\omega = N!$ and let $m \in T_i$ be an arbitrary element. It suffices to show that $m^\omega \equiv_i 1$, i.e., for all $x, y \in T_i$, we have $h_i(p)xm^\omega yh_i(q) = h_i(p)xyh_i(q)$.

Let $v_i, w_i \in A^*$ as in the definition of isolated occurrences and let $e = h(v_i^\omega)$ and $f = h(w_i^\omega)$. Note that $h_i(pv_i) = h_i(p)$ implies $h_i(p)e = h_i(p)$. Analogously, we have $fh_i(q) = h_i(q)$. Since B is contained in $\mathsf{alph}(v_i) = \mathsf{alph}(w_i)$ and since $m, x, y \in T_i = h_i(B^*)$, we have $e \mathrel{\mathcal{J}} f \leqslant_{\mathcal{J}} m^\omega, x, y$ by Lemma 19. Therefore,

$$h_i(p)xm^\omega yh_j(q) = h_i(p)exm^\omega yfh_i(q) = h_i(p)exyfh_i(q) = h_i(p)xyh_i(q)$$

where the second equality uses Lemma 21. ◀

We now return to the proof of the main theorem of this section.

Proof of Theorem 12. By considering a piecewise isolating factorization of w, it suffices to show that if u is an isolated occurrence in $w = puq$, then there exists an SLP S of polynomial size with $h_i(pSq) = h_i(puq)$ for all $i \in \{1,\ldots,k\}$. Combining the letters a_i and the SLPs for the isolated occurrences in the piecewise isolating factorization, we obtain the SLP for w.

Let again $B = \mathsf{alph}(u)$. To obtain a polynomial-size SLP S with $h_i(pSq) = h_i(puq)$ for all $i \in \{1,\ldots,k\}$, we consider the morphisms $\psi_i\colon B^* \to T_i/{\equiv_i}$ defined by $\psi_i(v) = [h_i(v)]_{\equiv_i}$, i.e., each word v is mapped to the equivalence class of $h_i(v)$ with respect to $\equiv_i$. Note that $|T_i/{\equiv_i}| \leqslant |T_i| \leqslant |M_i|$ for $1 \leqslant i \leqslant k$ and by Lemma 23, each of the monoids $T_i/{\equiv_i}$ is a group. By Theorem 18, there exists a polynomial-size SLP S with $\psi_i(S) = \psi_i(u)$ for all $i \in \{1,\ldots,k\}$ and, by the definition of $\equiv_i$, we obtain $h_i(pSq) = h_i(puq)$, as desired. ◀

6 Summary and Outlook

We investigated the complexity of the intersection problem for finite monoids, showing that the problem is NP-complete for varieties contained in **DO** and PSPACE-complete for varieties not contained within **DS**. To obtain a dichotomy result, one needs to investigate the complexity of the problem when monoids from $\mathbf{DS} \setminus \mathbf{DO}$ are part of the input. Using techniques similar to those in Section 5, we were able to show that for a subset of this class,

the problem remains NP-complete and thus, we conjecture that the problem is NP-complete whenever $\mathbf{V} \subseteq \mathbf{DS}$. The fact that $\mathbf{DS} \setminus \mathbf{DO}$ have not been studied and understood well enough from a language-theoretic perspective makes the problem of classifying the complexity of these monoids challenging but, at the same time, an interesting object for further research.

References

1. Jorge Almeida. *Finite Semigroups and Universal Algebra*. World Scientific, Singapore, 1994.
2. László Babai, Eugene M. Luks, and Ákos Seress. Permutation groups in NC. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 409–420. ACM, 1987. doi:10.1145/28395.28439.
3. László Babai, Eugene M. Luks, and Ákos Seress. Permutation groups in NC. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 409–420. ACM, 1987. doi:10.1145/28395.28439.
4. Martin Beaudry. Membership testing in commutative transformation semigroups. *Inf. Comput.*, 79(1):84–93, 1988. doi:10.1016/0890-5401(88)90018-1.
5. Martin Beaudry. *Membership Testing in Transformation Monoids*. PhD thesis, McGill University, Montreal, Quebec, 1988.
6. Martin Beaudry. Membership testing in threshold one transformation monoids. *Inf. Comput.*, 113(1):1–25, 1994. doi:10.1006/inco.1994.1062.
7. Martin Beaudry, Pierre McKenzie, and Denis Thérien. The membership problem in aperiodic transformation monoids. *J. ACM*, 39(3):599–616, 1992. doi:10.1145/146637.146661.
8. László Bernátsky. Regular expression star-freeness is PSPACE-complete. *Acta Cybernetica*, 13(1):1–21, 1997.
9. Michael Blondin, Andreas Krebs, and Pierre McKenzie. The complexity of intersecting finite automata having few final states. *Computational Complexity*, 25(4):775–814, 2016. doi:10.1007/s00037-014-0089-9.
10. Bogdan S. Chlebus. Domino-tiling games. *J. Comput. Syst. Sci.*, 32(3):374–392, 1986. doi:10.1016/0022-0000(86)90036-X.
11. Sung Cho and Dung T. Huynh. Finite automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88:96–116, 1991.
12. Volker Diekert, Claudio Gutiérrez, and Christian Hagenah. The existential theory of equations with rational constraints in free groups is PSPACE-complete. *Information and Computation*, 202:105–140, 2005. Conference version in STACS 2001, LNCS 2010, 170–182, 2004.
13. Merrick L. Furst, John E. Hopcroft, and Eugene M. Luks. Polynomial-time algorithms for permutation groups. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 36–41. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.34.
14. Markus Holzer and Martin Kutrib. Descriptional and computational complexity of finite automata - A survey. *Inf. Comput.*, 209(3):456–470, 2011. doi:10.1016/j.ic.2010.11.013.
15. Tao Jiang and Bala Ravikumar. Minimal NFA problems are hard. In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo, editors, *ICALP*, volume 510 of *Lecture Notes in Computer Science*, pages 629–640. Springer, 1991.
16. Dexter Kozen. Lower bounds for natural proof systems. In *Proc. of the 18th Ann. Symp. on Foundations of Computer Science, FOCS'77*, pages 254–266, Providence, Rhode Island, 1977. IEEE Computer Society Press.

17 Klaus-Jörn Lange and Peter Rossmanith. The emptiness problem for intersections of regular languages. In Ivan M. Havel and Václav Koubek, editors, *Mathematical Foundations of Computer Science 1992, 17th International Symposium, MFCS'92, Prague, Czechoslovakia, August 24-28, 1992, Proceedings*, volume 629 of *Lecture Notes in Computer Science*, pages 346–354. Springer, 1992. doi:10.1007/3-540-55808-X_33.

18 Jean-Éric Pin. *Varieties of Formal Languages*. North Oxford Academic, London, 1986.

19 Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

20 Pascal Tesson and Denis Thérien. Diamonds are forever: The variety DA. In Gracinda Maria dos Gomes Moreira da Cunha, Pedro Ventura Alves da Silva, and Jean-Éric Pin, editors, *Semigroups, Algorithms, Automata and Languages, Coimbra (Portugal) 2001*, pages 475–500. World Scientific, 2002.

Automata Theory on Sliding Windows

Moses Ganardi
Universität Siegen, Germany
ganardi@eti.uni-siegen.de

Danny Hucke
Universität Siegen, Germany
hucke@eti.uni-siegen.de

Daniel König
Universität Siegen, Germany
koenig@eti.uni-siegen.de

Markus Lohrey
Universität Siegen, Germany
lohrey@eti.uni-siegen.de

Konstantinos Mamouras
University of Pennsylvania, Philadelphia, USA
mamouras@seas.upenn.edu

Abstract

In a recent paper we analyzed the space complexity of streaming algorithms whose goal is to decide membership of a sliding window to a fixed language. For the class of regular languages we proved a space trichotomy theorem: for every regular language the optimal space bound is either constant, logarithmic or linear. In this paper we continue this line of research: We present natural characterizations for the constant and logarithmic space classes and establish tight relationships to the concept of language growth. We also analyze the space complexity with respect to automata size and prove almost matching lower and upper bounds. Finally, we consider the decision problem whether a language given by a DFA/NFA admits a sliding window algorithm using logarithmic/constant space.

2012 ACM Subject Classification Theory of computation → Streaming models, Theory of computation → Regular languages

Keywords and phrases Regular Languages, Sliding Window Algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.31

Related Version A full version of the paper with all proofs can be found at `https://arxiv.org/abs/1702.04376`, [19].

1 Introduction

Streaming algorithms process an input sequence $a_1 a_2 \cdots a_m$ from left to right and have at time t only direct access to the current data value a_t. Such algorithms have received a lot of attention in recent years; see [1] for an introduction. The general goal of streaming algorithms is to avoid the explicit storage of the whole data stream. Ideally, a streaming algorithm works in constant space, in which case it reduces to a deterministic finite automaton (DFA), but polylogarithmic space with respect to the input length might be acceptable, too. These small space requirements are motivated by the current explosion in the size of the input data, which makes random access to the input often infeasible. Such a scenario arises for instance

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 31; pp. 31:1–31:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

when searching in large databases (e.g., genome databases or web databases), analyzing internet traffic (e.g. click stream analysis), and monitoring networks.

The first papers on streaming algorithms are usually attributed to Munro and Paterson [28] and Flajolet and Martin [17], although the principle idea goes back to the work on online machines by Hartmanis, Lewis and Stearns from the 1960's [27, 31]. Extremely influential for the area of streaming algorithms was the paper of Alon, Matias, and Szegedy [2].

The sliding window model. Two streaming models can be found in the literature: In the *standard model* the algorithm reads a data stream $a_1 \cdots a_m$ from left to right. At time instant t it outputs a value $f(a_1 \cdots a_t)$ for a certain function f. In contrast, in the *sliding window model* the algorithm works on a sliding window. At time instant t, the active window is a certain suffix $a_{t-n+1} \cdots a_t$ of $a_1 \cdots a_t$ and the algorithm outputs $f(a_{t-n+1} \cdots a_t)$.

The sliding window model is the right approach for streaming applications, where data items are outdated after a certain time. A typical example is the analysis of a time series as it may arise in medical monitoring, web tracking, or financial monitoring. In all these applications, data items are usually no longer important after a certain time. Two variants of the sliding window model can be found in the literature; see e.g. [3]:

- *Fixed-size model:* The size of the active window is a fixed constant (the window size). In other words: at each time instant a new data value a_i arrives and the oldest data value from the sliding window expires.
- *Variable-size model:* The active window $a_{t-n+1} \cdots a_t$ is determined by an adversary. At every time instant the adversary can either remove the first data value from the window (expiration of a value) or add a new data value at the right end (arrival of a new value).

In the seminal paper of Datar et al. [15], where the fixed-size sliding window model was introduced, the authors show how to maintain the number of 1's in a sliding window of size n over the alphabet $\{0, 1\}$ in space $\frac{1}{\varepsilon} \cdot \log^2 n$ if one allows a multiplicative error of $1 \pm \varepsilon$. This is has been the starting point for a large number of further papers on the approximation of statistical data over sliding windows. Let us mention the work on computation of the variance and k-median [4], quantiles [3], and entropy [8] over sliding windows. Other computational problems that have been considered for the sliding window model include optimal sampling [9], various pattern matching problems [10, 11, 12, 13], database querying (e.g. processing of join queries [22]) and graph problems (e.g. checking for connectivity and computation of matchings, spanners, and spanning trees [14]). Further references on the sliding window model can be found in [1, Chapter 8] and [7].

Language recognition in the streaming model. A natural problem that has been surprisingly neglected for the streaming model is language recognition. The goal is to check whether an input string belongs to a given language L. Let us quote Magniez, Mathieu, and Nayak [26]: "Few applications [of streaming] have been made in the context of formal languages, which may have impact on massive data such as DNA sequences and large XML files. For instance, in the context of databases, properties decidable by streaming algorithm have been studied [30, 29], but only in the restricted case of deterministic and constant memory space algorithms." For Magniez et al. this was the starting point to study language recognition in the streaming model. Thereby they restricted their attention to the above mentioned standard streaming model. Note that in the standard model the membership problem for a regular language is trivial to solve: one simply simulates a DFA on the stream and thereby only store the current state. In [26] the authors present a randomized streaming algorithm for the (non-regular) Dyck language D_s with s pairs of parenthesis that works in space $\mathcal{O}(\sqrt{n} \log n)$

and time polylog(n) per symbol. Further investigations on streaming language recognition for various subclasses of context-free languages can be found in [5, 6, 18, 23, 24, 29, 30]. Let us emphasize that all these papers exclusively deal with the standard streaming model. Language recognition problems for the sliding window model have been completely neglected so far. This was the starting point for our previous paper [20].

Querying regular languages in the sliding window model. As mentioned above, in the standard streaming model the membership problem for a regular language can be solved in constant space by simulating a DFA. This solution does not work for the sliding window model. The problem is the removal of the left-most symbol from the sliding window. In order to check whether the active window belongs to a certain language L one has to know this first symbol in general. In such a case one has to store the whole window content using $\mathcal{O}(n)$ bits (where n is the window size). A simple regular language where this phenomenon arises is the language $a\{a,b\}^*$ of all words that start with a. The point is that by repeatedly checking whether the sliding window content belongs to $a\{a,b\}^*$, one can recover the exact content of the sliding window, which implies that every sliding window algorithm for testing membership in $a\{a,b\}^*$ has to use n bits of storage (where n is the window size).

For a function $s(n)$ let $\mathsf{F}_{\mathsf{reg}}(s(n))$ be the class of all languages L with the following property: For every window size n there exists an algorithm that reads a data stream, uses only space $s(n)$ and correctly decides at every time instant whether the active window (the last n symbols from the stream) belongs to L. Note that this is a *non-uniform* model: for every window size n we use a separate algorithm. The class $\mathsf{V}_{\mathsf{reg}}(s(n))$ of languages that have variable-size sliding window algorithms with space complexity $s(n)$ is defined similarly, see page 6 for details. Our main result from [20] is a space trichotomy for regular languages:

1. $\mathsf{V}_{\mathsf{reg}}(o(n)) = \mathsf{F}_{\mathsf{reg}}(o(n)) = \mathsf{F}_{\mathsf{reg}}(\mathcal{O}(\log n)) = \mathsf{V}_{\mathsf{reg}}(\mathcal{O}(\log n))$
2. $\mathsf{F}_{\mathsf{reg}}(o(\log n)) = \mathsf{F}_{\mathsf{reg}}(\mathcal{O}(1))$
3. $\mathsf{V}_{\mathsf{reg}}(o(\log n)) = \mathsf{V}_{\mathsf{reg}}(\mathcal{O}(1)) =$ all trivial languages (empty and universal languages)

Each of the three cases is characterized in terms of the syntactic homomorphism and the left Cayley graph of the syntactic monoid of the regular language. The precise characterizations are a bit technical; see [20] for details.

In this paper we continue our investigation of sliding-window algorithms for regular languages. As a first contribution, we present very natural characterizations of the above language classes in 1. and 2.: The languages in 1. are exactly the languages that are reducible with a Mealy machine (working from right to left) to a regular language of polynomial growth. The regular languages of polynomial growth are exactly the bounded regular languages [33]. A language L is *bounded* if $L \subseteq w_1^* w_2^* \cdots w_n^*$ for words $w_1, w_2, \ldots, w_n$. In addition, we show that the class 1. is the Boolean closure of regular left ideals (regular languages L with $\Sigma^* L \subseteq L$) and regular length languages (regular languages where $|u| = |v|$ implies that $u \in L$ iff $v \in L$). The class 2. is characterized as the Boolean closure of suffix-testable languages (languages L where membership in L only depends on a suffix of constant length) and regular length languages. A natural example for the classes above is the problem of testing whether the sliding window contains a fixed pattern w as a factor (or as a suffix) since we can check membership of the left ideal $\Sigma^* w \Sigma^*$ (or of the suffix-testable language $\Sigma^* w$).

We also consider the sliding-window space complexity of regular languages in a uniform setting, where the size m (number of states) of an automaton for the regular language is also taken into account. In [20], we asked whether for DFAs of size m that accept languages in $\mathsf{F}_{\mathsf{reg}}(\mathcal{O}(\log n)) = \mathsf{V}_{\mathsf{reg}}(\mathcal{O}(\log n))$, there exists a sliding-window streaming algorithm with space complexity $\mathrm{poly}(m) \cdot \log n$. Here, we give a negative answer by proving a lower bound

of the form $\Omega(2^m \cdot \log n)$. Moreover, we also show almost matching upper bounds.

Finally, we prove that one can test in nondeterministic logspace (NL) and hence in deterministic polynomial time whether for a given DFA $\mathcal{A}$ the language $L(\mathcal{A})$ belongs to the above class 1. (resp., 2.). For NFAs these problems become PSPACE-complete.

2 Preliminaries

For an alphabet Σ and $n \geq 0$ let $\Sigma^{\leq n} = \{x \in \Sigma^* : |x| \leq n\}$. The set of all *prefixes* of $x \in \Sigma^*$ is $\mathrm{Pref}(x) = \{u \in \Sigma^* \colon \exists v \in \Sigma^* : x = uv\}$ and the *reversal* of $x = a_1 \cdots a_n$ is $x^{\mathsf{R}} = a_n \cdots a_1$. For a language $L \subseteq \Sigma^*$ let $\mathrm{Pref}(L) = \bigcup_{x \in L} \mathrm{Pref}(x)$ and $L^{\mathsf{R}} = \{x^{\mathsf{R}} : x \in L\}$. The *reversal* of a function $\tau \colon \Sigma^* \to \Gamma^*$ is defined as $\tau^{\mathsf{R}}(x) = \tau(x^{\mathsf{R}})^{\mathsf{R}}$. Thus, $\tau(u) = v$ iff $\tau^{\mathsf{R}}(u^{\mathsf{R}}) = v^{\mathsf{R}}$.

We use $\log x$ as an abbreviation for $\lfloor \log_2 x \rfloor$. Note that if $(w_i)_{i \geq 0}$ is the length-lexicographic enumeration of $\{0,1\}^*$ then $|w_i| \leq \log i$. We use the following well-known bounds for binomial coefficients, where $1 \leq k \leq n$ and e is Euler's constant: $(n/k)^k \leq \binom{n}{k} \leq (e \cdot n/k)^k$.

We use standard definitions from automata theory. A *nondeterministic finite automaton* (NFA) is a tuple $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ where Q is a finite set of states, Σ is an alphabet, $I \subseteq Q$ is the set of initial states, $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation and $F \subseteq Q$ is the set of final states. A *deterministic finite automaton* (DFA) $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ has a single initial state $q_0 \in Q$ instead of I and a transition function $\delta \colon Q \times \Sigma \to Q$ instead of the transition relation Δ. A *deterministic automaton* has the same format as a DFA, except that the state set Q is not required to be finite. If $\mathcal{A}$ is deterministic, the transition function δ is extended to a function $\delta \colon Q \times \Sigma^* \to Q$ in the usual way and we define $\mathcal{A}(x) = \delta(q_0, x)$ for $x \in \Sigma^*$ and $L(\mathcal{A}) = \{x \in \Sigma^* \colon \mathcal{A}(x) \in F\}$ (the language accepted by $\mathcal{A}$).

Let $L \subseteq \Sigma^*$ be a language. The *left quotient* of $x \in \Sigma^*$ is $x^{-1}L = \{z \in \Sigma^* : xz \in L\}$. The *Myhill-Nerode congruence* $\sim_L$ is the equivalence relation on Σ^* defined by $x \sim_L y$ if and only if $x^{-1}L = y^{-1}L$. It is a *right congruence* on Σ^*, i.e. $x \sim_L y$ implies $xz \sim_L yz$ for all $x, y, z \in \Sigma^*$. If $\mathcal{A}$ is a (not necessarily finite) deterministic automaton for a language $L \subseteq \Sigma^*$, then $\mathcal{A}(x) = \mathcal{A}(y)$ implies $x \sim_L y$. The *minimal deterministic automaton* for L is $\mathcal{A}_L = (\Sigma^*/\!\sim_L, \Sigma, [\varepsilon]_{\sim_L}, \delta, \{[x]_{\sim_L} : x \in L\})$ with $\delta([x]_{\sim_L}, a) = [xa]_{\sim_L}$. Clearly, $L(\mathcal{A}_L) = L$.

For an NFA $\mathcal{A}$ we denote with $\mathcal{A}^{\mathsf{D}}$ the corresponding deterministic power set automaton (restricted to those states that are reachable from the initial state) and with $\mathcal{A}^{\mathsf{R}}$ the NFA obtained from $\mathcal{A}$ by reversing all transitions and swapping the set of initial states and the set of final states. Moreover, we define $\mathcal{A}^{\mathsf{RD}} = (\mathcal{A}^{\mathsf{R}})^{\mathsf{D}}$. Thus, $L(\mathcal{A}^{\mathsf{R}}) = L(\mathcal{A}^{\mathsf{RD}}) = L(\mathcal{A})^{\mathsf{R}}$. If an NFA $\mathcal{A}$ has m states, then both $\mathcal{A}^{\mathsf{D}}$ and $\mathcal{A}^{\mathsf{RD}}$ have at most 2^m states.

3 Streaming algorithms

A data stream is just a finite sequence of data values. We make the assumption that these data values are from a finite set Σ. Thus, a data stream is a finite word $w = a_1 \cdots a_m \in \Sigma^*$. A streaming algorithm reads the symbols of a data stream from left to right. At time instant t the algorithm has only access to the symbol a_t and the internal storage, which is encoded by a bit string. The goal of the streaming algorithm is to compute a certain function $f \colon \Sigma^* \to A$ into some domain A, which means that at time instant t the streaming algorithm outputs the value $f(a_1 \cdots a_t)$. In this paper, we only consider the Boolean case $A = \{0,1\}$; in other words, the streaming algorithm tests membership in a fixed language. Furthermore, we abstract away from the actual computation and only analyze the space requirement. Formally, a *streaming algorithm* for $L \subseteq \Sigma^*$ is a deterministic (possibly infinite) automaton $\mathcal{A} = (S, \Sigma, s_0, \delta, F)$ with $L = L(\mathcal{A})$, where the states are encoded by bit strings.

We describe this encoding by an injective function enc: $S \to \{0,1\}^*$. The *space function* space$(\mathcal{A}, \cdot)$: $\Sigma^* \to \mathbb{N}$ specifies the space used by $\mathcal{A}$ on a certain input: For $w \in \Sigma^*$ let space$(\mathcal{A}, w) = \max\{|\text{enc}(\mathcal{A}(u))| : u \in \text{Pref}(w)\}$.

In the above streaming model, the output value of the streaming algorithm at time t depends on the whole past $a_1 a_2 \cdots a_t$ of the data stream. However, in many practical applications one is only interested in the "relevant part of the past". Two formalizations of this can be found in the literature:

- Only the suffix of $a_1 a_2 \cdots a_t$ of length n is relevant. Here, n is a fixed constant. This streaming model is called the *fixed-size sliding window model*.
- The relevant suffix of $a_1 a_2 \cdots a_t$ is determined by an adversary. In this model, at every time instant the adversary can either remove the first symbol from the active window (expiration of a data value), or add a new symbol at the right end (arrival of a new data value). This streaming model is also called the *variable-size sliding window model*.

Fixed-size sliding windows. Given a word $w = a_1 a_2 \cdots a_m \in \Sigma^*$ and a window length $n \geq 0$, we define the *active window* $\text{last}_n(w) = a_{m-n+1} a_{m-n+2} \cdots a_m \in \Sigma^n$, where we set $a_i = a$ for $i \leq 0$. Here $a \in \Sigma$ is an arbitrary symbol, which fills the window initially. A sequence $\mathcal{A} = (\mathcal{A}_n)_{n \geq 0}$ is a *fixed-size sliding window algorithm* for $L \subseteq \Sigma^*$ if each $\mathcal{A}_n$ is a streaming algorithm for $\{w \in \Sigma^* : \text{last}_n(w) \in L\}$. Its *space complexity* is the function $f_{\mathcal{A}}: \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ where $f_{\mathcal{A}}(n)$ is the maximum encoding length of a state in $\mathcal{A}_n$.

Note that for every language L and every n the language $\{w \in \Sigma^* : \text{last}_n(w) \in L\}$ is regular, which ensures that $\mathcal{A}_n$ can be chosen to be a DFA and hence $f_{\mathcal{A}}(n) < \infty$ for all $n \geq 0$. The trivial fixed-size sliding window algorithm for L is the sequence $\mathcal{B} = (\mathcal{B}_n)_{n \geq 0}$, where $\mathcal{B}_n$ is the DFA with state set Σ^n and the transition mapping $\delta(au, b) = ub$ for $a, b \in \Sigma$, $u \in \Sigma^{n-1}$. States of $\mathcal{B}_n$ can be encoded with $\mathcal{O}(\log |\Sigma| \cdot n)$ bits. By minimizing each $\mathcal{B}_n$, we obtain an *optimal fixed-size sliding window algorithm* $\mathcal{A}$ for L. Finally, we define $F_L(n) = f_{\mathcal{A}}(n)$. Thus, F_L is the space complexity of an optimal fixed-size sliding window algorithm for L. Notice that F_L is not necessarily monotonic. For instance, take $L = \{au \colon u \in \{a,b\}^*, |u| \text{ odd}\}$. Then, we have $F_L(2n) \in \Theta(n)$ and $F_L(2n+1) \in O(1)$. The above trivial algorithm $\mathcal{B}$ yields $F_L(n) \in \mathcal{O}(n)$ for every language L.

Note that the fixed-size sliding window is a *non-uniform* model: for every window size we have a separate streaming algorithm and these algorithms do not have to follow a common pattern. Working with a non-uniform model makes lower bounds stronger. In contrast, the variable-size sliding window model that we discuss next is a uniform model in the sense that there is a single streaming algorithm that works for every window length.

Variable-size sliding windows. For an alphabet Σ we define the extended alphabet $\overline{\Sigma} = \Sigma \cup \{\downarrow\}$. In the variable-size model the *active window* $\text{wnd}(u) \in \Sigma^*$ for a stream $u \in \overline{\Sigma}^*$ is defined as follows, where $a \in \Sigma$:

$$\begin{aligned} \text{wnd}(\varepsilon) &= \varepsilon & \text{wnd}(u\downarrow) &= \varepsilon \text{ if } \text{wnd}(u) = \varepsilon \\ \text{wnd}(ua) &= \text{wnd}(u)a & \text{wnd}(u\downarrow) &= v \text{ if } \text{wnd}(u) = av \end{aligned}$$

A *variable-size sliding window algorithm* for a language $L \subseteq \Sigma^*$ is a streaming algorithm $\mathcal{A}$ for $\{w \in \overline{\Sigma}^* : \text{wnd}(w) \in L\}$. Its *space complexity* is the function $v_{\mathcal{A}}$ mapping a window length n to the maximum number of bits used by $\mathcal{A}$ on inputs producing an active window of size at most n. Formally, it is the monotonic function $v_{\mathcal{A}}: \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ with $v_{\mathcal{A}}(n) = \max\{\text{space}(\mathcal{A}, u) : u \in \overline{\Sigma}^*, |\text{wnd}(v)| \leq n \text{ for all } v \in \text{Pref}(u)\}$. This definition of $v_{\mathcal{A}}(n)$ slightly deviates from the one given in [20], where the space complexity is defined as $v'_{\mathcal{A}}(n) =$

$\max\{|\mathrm{enc}(\mathcal{A}(u))| : u \in \overline{\Sigma}^*, |\mathrm{wnd}(u)| = n\}$. One easily sees that $v_{\mathcal{A}}(n) = \max_{k \le n} v'_{\mathcal{A}}(k)$ and hence $v_{\mathcal{A}}(n) = v'_{\mathcal{A}}(n)$ if $v'_{\mathcal{A}}(n)$ is monotonic. An advantage of our definition of $v_{\mathcal{A}}(n)$ is that for every language an optimal variable-size sliding window algorithm exists. We obtain this algorithm from the minimal deterministic automaton for $\{w \in \overline{\Sigma}^* : \mathrm{wnd}(w) \in L\}$.

▶ **Lemma 3.1.** *For every $L \subseteq \Sigma^*$ there exists a variable-size sliding window algorithm $\mathcal{A}$ such that $v_{\mathcal{A}}(n) \le v_{\mathcal{B}}(n)$ for every variable-size sliding window algorithm $\mathcal{B}$ for L and all n.*

We define $V_L(n) = v_{\mathcal{A}}(n)$, where $\mathcal{A}$ is a space *optimal variable-size sliding window algorithm* for L from Lemma 3.1. Since any algorithm in the variable-size model yields an algorithm in the fixed-size model, we have $F_L(n) \le V_L(n)$.

Space complexity classes and closure properties. For a function $s\colon \mathbb{N} \to \mathbb{N}$ we define the classes $\mathsf{F}(s)$ and $\mathsf{V}(s)$ of all languages $L \subseteq \Sigma^*$ which have a fixed-size (variable-size, respectively) sliding window algorithm with space complexity bounded by $s(n)$. For a class $\mathcal{C}$ of functions we define $\mathsf{X}(\mathcal{C}) = \bigcup_{s \in \mathcal{C}} \mathsf{X}(s)$ for $\mathsf{X} \in \{\mathsf{F}, \mathsf{V}\}$.

Several times we will exploit closure properties of the classes $\mathsf{F}(\mathcal{O}(s))$ and $\mathsf{V}(\mathcal{O}(s))$ (for a function $s(n)$). We need the following definitions: A *Mealy machine* $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta)$ consists of a finite set of states Q, an input alphabet Σ, an output alphabet Γ, an initial state $q_0 \in Q$ and the transition function $\delta\colon Q \times \Sigma \to Q \times \Gamma$. For every $q \in Q$ the machine computes a length-preserving transduction $\tau_q\colon \Sigma^* \to \Gamma^*$ in the usual way: $\tau_q(\varepsilon) = \varepsilon$ and if $\delta(p, a) = (q, b)$ then $\tau_p(au) = b\,\tau_q(u)$. We call $\tau^{\mathsf{R}}_{q_0}$ the *$\leftarrow$-transduction* computed by $\mathcal{M}$. Thus, a $\leftarrow$-transduction is computed by a Mealy machine that works on an input word from right to left. If L is regular and τ is a $\leftarrow$-transduction, then $\tau(L)$ and $\tau^{-1}(L)$ are regular as well. A $\leftarrow$-transduction τ is called a *$\leftarrow$-reduction* from $K \subseteq \Sigma^*$ to $L \subseteq \Gamma^*$ if $K = \tau^{-1}(L)$.

▶ **Lemma 3.2.** *For any function $s(n)$ the classes $\mathsf{F}(\mathcal{O}(s))$ and $\mathsf{V}(\mathcal{O}(s))$ are closed under (i) Boolean operations and (ii) $\leftarrow$-reductions.*

Space trichotomy for regular languages. In [20] we proved a trichotomy theorem on sliding window algorithms for regular languages. We identified a partition of the class of regular languages into three classes which completely characterize the sliding window space complexity in both the fixed-size and the variable-size model. The definition of the three classes is given in terms of the syntactic homomorphism and the left Cayley graph of the syntactic monoid of the regular language, see [20].

For $\mathsf{X} \in \{\mathsf{F}, \mathsf{V}\}$ and a class $\mathcal{C}$ of functions we abbreviate $\mathsf{X}(\mathcal{C}) \cap \mathsf{REG}$ by $\mathsf{X}_{\mathsf{reg}}(\mathcal{C})$, where REG is the class of all regular languages.

▶ **Theorem 3.3** ([20])**.** *The following holds:*

- $\mathsf{V}_{\mathsf{reg}}(o(n)) = \mathsf{F}_{\mathsf{reg}}(o(n)) = \mathsf{F}_{\mathsf{reg}}(\mathcal{O}(\log n)) = \mathsf{V}_{\mathsf{reg}}(\mathcal{O}(\log n))$
- $\mathsf{F}_{\mathsf{reg}}(o(\log n)) = \mathsf{F}_{\mathsf{reg}}(\mathcal{O}(1))$
- $\mathsf{V}_{\mathsf{reg}}(o(\log n)) = \mathsf{V}_{\mathsf{reg}}(\mathcal{O}(1)) =$ *all trivial languages (empty and universal languages)*

Strictly speaking, [20, Theorem 7] only claims $\mathsf{V}_L(n) \notin O(1)$ for all non-trivial languages L. However, the proof of [20, Theorem 7] does imply the stronger bound $\mathsf{V}_L(n) \notin o(\log n)$. This statement will also be reproved in the following section.

Let us comment on a subtle point. When making statements about the space complexity functions $V_L(n)$ and $F_L(n)$ it is in general important to fix the underlying alphabet. For instance according to the third point from Theorem 3.3 we have $V_L(n) \in \mathcal{O}(1)$ for the language $L = \{a\}^*$ if the underlying alphabet is $\{a\}$. On the other hand, if the underlying alphabet is $\{a, b\}$ then $V_L(n) \notin \mathcal{O}(1)$ (in fact, L then belongs to $\mathsf{V}_{\mathsf{reg}}(\Theta(\log n))$).

4 Space complexity and language growth

In this section we reprove the space trichotomy (Theorem 3.3) for the variable-size model. For this we relate the function $V_L(n)$ to the growth of a certain derived language and then use the well known results about the growth of regular languages. We need the following definition. For a language $L \subseteq \Sigma^*$ define the mapping $\psi_L \colon \Sigma^* \to (\Sigma^*/\sim_L)^*$ by $\psi_L(a_1 \cdots a_n) = [a_1 \cdots a_n]_{\sim_L}[a_2 \cdots a_n]_{\sim_L} \cdots [a_n]_{\sim_L}$. Notice that ψ_L is a length-preserving mapping from Σ^* to the set of words over the alphabet $\Sigma^*/\sim_L$. Although $\Sigma^*/\sim_L$ may be infinite (namely for non-regular L), the image $\psi_L(\Sigma^{\leq n})$ has at most $|\Sigma|^{n+1} - 1$ elements for each $n \geq 0$.

▶ **Theorem 4.1.** *For every language $\emptyset \subsetneq L \subsetneq \Sigma^*$ we have $V_L(n) = \log |\psi_L(\Sigma^{\leq n})|$.*

Proof sketch. We first exhibit a variable-size sliding window algorithm $\mathcal{A}$ with $v_{\mathcal{A}}(n) = \log |\psi_L(\Sigma^{\leq n})|$. The idea is that on input $w \in \overline{\Sigma}^*$ the algorithm $\mathcal{A}$ is in state $\mathcal{A}(w) = \psi_L(\mathrm{wnd}(w))$. Consider an active window $a_1 \cdots a_n \in \Sigma^*$. Three observations are crucial:

- $\psi_L(a_2 \cdots a_n)$ is obtained from $\psi_L(a_1 \cdots a_n)$ by removing the first $\sim_L$-class $[a_1 \cdots a_n]_{\sim_L}$.
- $\psi_L(a_1 \cdots a_n)$ and $a \in \Sigma$ determine $\psi_L(a_1 \cdots a_n a) = [a_1 \cdots a_n a]_{\sim_L} \cdots [a_n a]_{\sim_L}[a]_{\sim_L}$, since $\sim_L$ is a right-congruence
- The first $\sim_L$-class in $\psi_L(a_1 \cdots a_n)$ determines whether $a_1 \cdots a_n \in L$.

These remarks define a variable-size sliding window algorithm for L with state set $\psi_L(\Sigma^*)$. It is easy to define a binary encoding of the states such that this variable-size sliding window algorithm has space complexity $\log |\psi_L(\Sigma^{\leq n})|$.

Conversely, consider a variable-size sliding window algorithm $\mathcal{A}$ for L with space complexity $v(n) = v_{\mathcal{A}}(n)$. To prove that $v(n) \geq \log |\psi_L(\Sigma^{\leq n})|$, one shows that for every input $x = a_1 a_2 \cdots a_m \in \Sigma^*$ of length $m \leq n$, the state $\mathcal{A}(x)$ determines (i) $m = |x|$ and (ii) $\psi_L(a_1 \cdots a_m)$. Hence, every value $\psi_L(x)$ for $x \in \Sigma^{\leq n}$ can be encoded by a bit string of length at most $v(n)$, namely $\mathrm{enc}(\mathcal{A}(x))$. Since there are $|\psi_L(\Sigma^{\leq n})|$ such values, it follows that $2^{v(n)+1} - 1 \geq |\psi_L(\Sigma^{\leq n})|$, which implies $v(n) \geq \log |\psi_L(\Sigma^{\leq n})|$. ◀

Lemma 4.1 fails for $L = \emptyset$ or $L = \Sigma^*$, where $V_L(n) = 0$ and $\log |\psi_L(\Sigma^{\leq n})| = \log(n+1)$.

We can use Lemma 4.1 to reprove the space trichotomy for regular languages in the variable-size sliding window model. For this, we need the following simple lemma:

▶ **Lemma 4.2.** *If $L \subseteq \Sigma^*$ is regular, then ψ_L is a $\leftarrow$-transduction. In particular, $\psi_L(\Sigma^*)$ and $\psi_L(L)$ are regular. Furthermore ψ_L is a $\leftarrow$-reduction from L to $\psi_L(L)$.*

The *growth* of a language $L \subseteq \Sigma^*$ is the function $g(n) = |\{x \in L : |x| \leq n\}|$. Since the growth of every regular language is either $\Theta(n^d)$ for some integer $d \geq 0$ or $\Omega(r^n)$ for some $r > 1$ [21, Section 2.3], Lemma 4.1 and 4.2 reprove the trichotomy theorem for variable-size windows: For a regular language L, $V_L(n)$ is either in $\mathcal{O}(1)$, $\Theta(\log n)$ or $\Theta(n)$. Furthermore, since $|\psi_L(\Sigma^{\leq n})| \geq n + 1$ we have $V_L(n) \in \Omega(\log n)$ for every non-trivial language L.

Let us conclude this section with a result that bounds for all languages the fixed-size space function $F_L(n)$ in terms of the growth of L.

▶ **Theorem 4.3.** *If $L \subseteq \Sigma^*$ has growth $g(n)$, then $F_L(n) \in \mathcal{O}(\log g(n) + \log n)$.*

5 Logspace sliding-window algorithms

In this section we will study the class $\mathsf{V}_{\mathsf{reg}}(\mathcal{O}(\log n))$. We will (i) give several new and very natural characterizations of $\mathsf{V}_{\mathsf{reg}}(\mathcal{O}(\log n))$, (ii) will exhibit a new logspace sliding-window algorithm that is more space efficient in terms of the automata size compared to our previous

algorithm from [20], and (iii) will match our new space bound by an almost tight lower bound. Before we state the results, we have to introduce a couple of definitions.

A *strongly connected component* (SCC for short) of a DFA $\mathcal{B} = (Q, \Sigma, q_0, \delta, F)$ is an inclusion-maximal subset $C \subseteq Q$ such that for all $p, q \in C$ there exist words $u, v \in \Sigma^*$ such that $\delta(p, u) = q$ and $\delta(q, v) = p$. An SCC $C \subseteq Q$ is *well-behaved* if for all $q \in C$ and $u, v \in \Sigma^*$ with $|u| = |v|$ and $\delta(q, u), \delta(q, v) \in C$ we have: $\delta(q, u) \in F$ if and only if $\delta(q, v) \in F$. If every SCC in $\mathcal{B}$ which is reachable from q_0 is well-behaved, then $\mathcal{B}$ is called *well-behaved*.

A language $L \subseteq \Sigma^*$ is called a *left ideal (right ideal)* if $\Sigma^* L \subseteq L$ ($L\Sigma^* \subseteq L$). A language $L \subseteq \Sigma^*$ is called a *length language* if for all $n \in \mathbb{N}$, either $\Sigma^n \subseteq L$ or $L \cap \Sigma^n = \emptyset$. Clearly, L is a length language iff L^{R} is a length language, and L is left ideal iff L^{R} is a right ideal. In this section we prove the main characterization theorem for the class $\mathsf{V}_{\mathsf{reg}}(\mathcal{O}(\log n))$:

▶ **Theorem 5.1.** *Let $L \subseteq \Sigma^*$ be regular. The following statements are equivalent:*

1. $L \in \mathsf{F}(\mathcal{O}(\log n))$
2. $L \in \mathsf{V}(\mathcal{O}(\log n))$
3. L^{R} *is recognized by a well-behaved DFA.*
4. L *is $\leftarrow$-reducible to a regular language of polynomial growth.*
5. L *is a Boolean combination of regular left ideals and regular length languages.*

Our proof of the direction from 3. to 2. will also yield a better space bound in terms of automata size. In [20] we presented a variable-size sliding window algorithm using space $\mathcal{O}(m^m \cdot (m \cdot \log(m) + \log(n)))$ for a regular language that is given by a DFA with m states.

▶ **Theorem 5.2.** *Let $\mathcal{A}$ be a DFA or NFA with m states such that $\mathcal{A}^{\mathsf{RD}}$ is well-behaved. There are constants c_m, d_m that only depend on m such that the following holds for $L = L(\mathcal{A})$:*

- *If $\mathcal{A}$ is a DFA then $V_L(n) \leq (2^m \cdot m + 1) \cdot \log n \; + \; c_m$ for n large enough.*
- *If $\mathcal{A}$ is an NFA then $V_L(n) \leq (4^m + 1) \cdot \log n \; + \; d_m$ for n large enough.*

Finally we prove a lower bound for the fixed-size model (and hence also for the variable-size model) that almost matches the space bound in Theorem 5.2:

▶ **Theorem 5.3.** *For all $k \geq 1$ there exists a language $L_k \subseteq \{0, \ldots, k\}^*$ recognized by a DFA with $k + 3$ states such that $L_k \in \mathsf{F}(\mathcal{O}(\log n))$ and $F_{L_k}(n) \geq (2^k - 1) \cdot (\log n - k)$.*

We start with the proof of Theorem 5.2.

5.1 Proof of Theorem 5.2

We need one more definition for the proof of Theorem 5.2. Let $\mathcal{B}$ be a well-behaved DFA with m states and let ρ be a run in $\mathcal{B}$, which does not necessarily start in the initial state. Let $C_1, \ldots, C_k$ be the sequence of pairwise different SCCs that are visited by ρ in that order. The *path summary* of ρ is the sequence $S(\rho) = (p_1, \ell_1, p_2, \ell_2, \ldots, p_k, \ell_k)$ where p_i is the first state in C_i visited by ρ, and $\ell_i \geq 0$ is the number of symbols read in ρ from the first occurrence of p_i until the first state from C_{i+1} (or until the end for p_k). The number of different path summaries $S(\rho)$, where ρ ranges over all runs in $\mathcal{B}$ of length n can be bounded by (e is Euler's constant)

$$m^m \cdot \binom{n+m-1}{m-1} \leq m^m \cdot \binom{n+m}{m} \leq m^m \cdot \left(\frac{e \cdot (n+m)}{m}\right)^m \leq e^m \cdot (n+m)^m. \qquad (1)$$

Here, (i) m^m is the number of sequences of m states (we can repeat the last state in a path summary so that we have exactly m states) and (ii) $\binom{n+m-1}{m-1}$ is the number of ordered partitions of n into m summands.

We can now prove Theorem 5.2. Let $L \subseteq \Sigma^*$ be regular and given by a finite DFA or NFA $\mathcal{A}$. Let $\mathcal{B} = \mathcal{A}^{\mathsf{RD}}$, which is well-behaved. A set $D \subseteq \Sigma^*$ *distinguishes* L if for all $x, y \in \Sigma^*$ with $x \not\sim_L y$ there exists $z \in D$ such that exactly one of the words xz and yz belongs to L. If $\mathcal{A}$ is a DFA with m states, then there are at most m distinct left quotients $x^{-1}L$. Since every family of m sets has a distinguishing set of size at most $m-1$ [16], we get a set D of size at most $m-1$ that distinguishes L. If $\mathcal{A}$ is an NFA with m states, we can clearly choose $|D| \leq 2^m - 1$ by determinizing $\mathcal{A}$.

For a window content $w = a_1 \cdots a_n$ we define a 0-1-matrix $A_w \colon D \times \{1, \ldots, n\} \to \{0,1\}$ by $A_w(z,i) = 1$ iff $a_i \cdots a_n z \in L$. Note that the i-th column $A_w(\cdot, i)$ determines $[a_i \cdots a_n]_{\sim_L}$, and vice versa. Hence, the matrix A_w determines $\psi_L(w)$ and vice versa, i.e., $|\psi_L(\Sigma^{\leq n})| = |\{A_w \colon w \in \Sigma^{\leq n}\}|$. By Lemma 4.1, it therefore suffices to bound $|\{A_w \colon w \in \Sigma^{\leq n}\}|$.

We can encode each row $A_w(z, \cdot)$ of A_w succinctly as follows. Consider one row indexed by $z \in D$. Let ρ_z be the run of $\mathcal{B}$ on the word $(wz)^{\mathsf{R}}$ and $\tilde{\rho}_z$ be the subrun of ρ_z which only reads the suffix w^{R} of $(wz)^{\mathsf{R}}$. One can reconstruct $A_w(z, \cdot)$ from the path summary $S(\tilde{\rho}_z)$. Thus A_w can be encoded by $|D|$ many path summaries. With (1) and the fact that $\mathcal{B}$ has at most 2^m states, we get the bound

$$|\{A_w \colon w \in \Sigma^{\leq n}\}| \leq \sum_{i=0}^{n} e^{2^m |D|} \cdot (i + 2^m)^{2^m |D|} \leq (n+1) \cdot e^{2^m |D|} \cdot (n + 2^m)^{2^m |D|}.$$

Hence, for the DFA case (where $|D| \leq m-1$) we have

$$V_L(n) = \log |\psi_L(\Sigma^{\leq n})| \leq \log(n+1) + 2^m \cdot m \cdot (\log e + \log(n + 2^m)) \leq (2^m \cdot m + 1) \cdot \log n \; + \; c_m$$

for n large enough, where c_m can be chosen as $1 + 2^m \cdot m \cdot \log e + m^2 \cdot 2^m$. The calculation for the NFA case (where $|D| \leq 2^m - 1$) is analogous.

5.2 Proof of Theorem 5.1

The equivalence of 1. and 2. is shown in [20] (its the only direction that we do not reprove), and the direction from 3. to 2. is stated in Theorem 5.2. The implication from 2. to 4. follows from Lemma 4.1 and 4.2. The direction from 2. to 3. is shown in the full version [19], where we actually show that $V_L(n) \in \Omega(n)$ if L^{R} is recognized by a DFA that is not well-behaved. To prove that 5. implies 3. we show in [19] that (i) the minimal DFA for a regular right ideal or a regular length language is well-behaved, and (ii) that the class of languages accepted by well-behaved DFAs is closed under Boolean operations.

It remains to show the implication from 4. to 5. First, a straightforward argument shows that the class of Boolean combinations of regular left ideals and regular length languages is closed under pre-images of $\leftarrow$-transductions (see [19]). Therefore, it suffices to prove that every regular language of polynomial growth is a Boolean combination of regular left ideals and regular length languages. Since a language L and its reversal L^{R} have the same growth, we can instead show that every regular language of polynomial growth is a Boolean combination of regular right ideals and regular length languages. The idea is to decompose every regular language of polynomial growth as a finite union of languages recognized by so called linear cycle automata.

In the following we will allow *partial* DFAs $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ where $\delta \colon Q \times \Sigma \to Q$ is a partial function. An SCC C of a partial DFA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ is called a *cycle* if for every $p \in C$ there exists at most one $a \in \Sigma$ such that $\delta(p, a) \in C$. Note that a singleton SCC $C = \{p\}$ such that $\delta(p, a) \neq p$ whenever $\delta(p, a)$ is defined is a cycle, too. Such a cycle is called *trivial*. A partial DFA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ is a *linear cycle automaton* if

- for all $p, q \in Q$ there exists at most one symbol $a \in \Sigma$ such that $\delta(p, a) = q$,

- every SCC C of $\mathcal{A}$ is a (possibly trivial) cycle,
- there is an enumeration $C_1, \ldots, C_k$ of the SCCs of $\mathcal{A}$ such that there is a unique transition from C_i to C_{i+1} for $1 \leq i \leq k-1$, and there is no transition from C_i to C_j for $j > i+1$,
- q_0 belongs to C_1,
- $|F| = 1$ and the unique final state belongs to C_k.

▶ **Lemma 5.4.** *If L is a regular language with polynomial growth, then L is a finite union of languages recognized by linear cycle automata.*

Proof. Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ be the minimal DFA for a regular language $L \subseteq \Sigma^*$ of polynomial growth. We first remove from $\mathcal{A}$ all states from which no state in F is reachable; then $\mathcal{A}$ becomes a partial DFA. By [21, Lemma 2] for every $q \in Q$ there exists a word $u_q \in \Sigma^*$ such that the language $\{w \in \Sigma^* : \delta(q, w) = q\}$ is a subset of u_q^*. Thus, for every SCC C of $\mathcal{A}$ and every state $q \in C$ there is at most one symbol $a \in \Sigma$ with $\delta(q, a) \in C$.

A *path description* is a sequence $P = (q_0, C_0, p_0, a_0, q_1, C_1, p_1, a_2, \ldots, q_k, C_k, p_k)$ where $C_0, \ldots, C_k$ is a chain in the partial ordering on the SCCs of $\mathcal{A}$, $q_i, p_i \in C_i$ for all $0 \leq i \leq k$, $\delta(p_i, a_i) = q_{i+1}$ for all $0 \leq i < k$ and $p_k \in F$. There are only finitely many path descriptions. To every accepting run of $\mathcal{A}$ we assign a path description, which indicates the SCCs traversed in the run and the transitions that lead from one SCC to the next SCC. We can write $L(\mathcal{A})$ as a finite union of languages over all path descriptions. For every path description P, we take the set of all words accepted by a run of $\mathcal{A}$ whose path description is P.

Consider a single path description $P = (q_0, C_0, p_0, a_0, q_1, C_1, p_1, a_2, \ldots, q_k, C_k, p_k)$ and let $\mathcal{B}$ be the restriction of $\mathcal{A}$ to the SCCs C_i. Furthermore all transitions between two distinct SCCs are removed except for the transitions (p_i, a_i, q_{i+1}). Finally, p_k becomes the only final state of $\mathcal{B}$. Then $\mathcal{B}$ is indeed a linear cycle automaton. ◀

By the previous lemma, it suffices to decompose the language accepted by a linear cycle automaton as a Boolean combination of regular length languages and regular right ideals. By a simple pumping argument (see [19]) we can reduce to linear cycle automata, in which each cycle has the same length. Let us consider such an automaton $\mathcal{A}$ and let $L = L(\mathcal{A})$: There are numbers $p, q \geq 0$ such that each word in L has length $p + qn$ for some $n \geq 0$. Here q is the uniform length of the non-trivial cycles in $\mathcal{A}$. We claim that L is the intersection of the three languages

- $L\Sigma^*$, which is a regular right ideal,
- $\{x \in \Sigma^* : \mathrm{Pref}(x) \subseteq \mathrm{Pref}(L)\}$, which is the complement of a regular right ideal,
- $\Sigma^p(\Sigma^q)^*$, which is a length language.

Clearly L is contained in the described intersection. Conversely, consider a word x in the intersection. We have $x = yz$ where $y \in L$. Hence, $|y| = p + qn$ for some n. Since $|x| = p + qn'$ for some n', the length $|z|$ is divided by q. Since $y \in L$, $\mathcal{A}(y)$ is the unique final state of $\mathcal{A}$, which belongs to the unique maximal SCC C of $\mathcal{A}$. If C is non-trivial, then it is a cycle of length q and also $\mathcal{A}(yz)$ is the final state, i.e., $x \in L$. If C is trivial, then $y, yz \in L$ implies $z = \varepsilon$ and x is also accepted by $\mathcal{A}$. This concludes the proof for the direction from 4. to 5.

5.3 Proof of Theorem 5.3

The languages L_k ($k \geq 0$) from Theorem 5.3 are defined as

$$L_0 = 0^+ \quad \text{and} \quad L_k = L_{k-1} \cup L_{k-1}\, k\, \{0, \ldots, k-1\}^* \text{ for } k \geq 1.$$

Observe that a word $a_1 \cdots a_n \in \{0, \ldots, k\}^*$ belongs to L_k if and only if $n \geq 1$, $a_1 = 0$ and for each $1 \leq i \leq n$ it holds that $a_i = 0$ or $a_i \neq \max_{1 \leq j \leq i-1} a_j$. We can construct a DFA $\mathcal{A}_k$ for L_k with $k + 3$ states, which stores the maximum value seen so far in its state, see Figure 1.

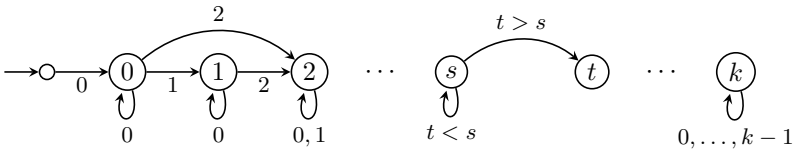


Figure 1 A DFA for L_k. Omitted transitions lead to a sink state. All non-sink states are final.

To prove that each L_k belongs to $\mathsf{V}(\mathcal{O}(\log n))$, we show that L_k is a Boolean combination of regular left ideals. Given a word $x = a_1 \cdots a_n \in \Sigma^*$ and a language $L \subseteq \Sigma^*$, a position $1 \leq i \leq n$ is an *L-alternation point*, if exactly one of the words $a_i \cdots a_n$ and $a_{i+1} \cdots a_n$ belongs to L. Denote by $\mathrm{alt}_L(x)$ the number of L-alternation points in x. We need the following two lemmas, which are proven in the full version [19].

▶ **Lemma 5.5.** *Let $L \subseteq \Sigma^*$ be regular. Then L is a Boolean combination of at most k regular left ideals if and only if $\mathrm{alt}_L(x) \leq k$ for all $x \in \Sigma^*$.*

▶ **Lemma 5.6.** *For all $k \geq 0$ and $x \in \mathbb{N}^*$ we have $\mathrm{alt}_{L_k}(x) \leq 2^{k+2} - 2$. Moreover, $V_{L_k}(n) \leq (2^{k+3} \cdot (k+3) + 1) \cdot \log n \; + \; c_k$ for n large enough, where c_k only depends on k.*

For the proof of the if-direction in Lemma 5.5, one writes L as a Boolean combination of the sets $\{x \in \Sigma^* : \mathrm{alt}_L(x) \geq i\}$ $(1 \leq i \leq k)$. Lemma 5.6 is shown by induction on k.

We can now prove Theorem 5.3. Define the languages $Z_0 = 0^*$ and $Z_k = Z_{k-1}\, k\, Z_{k-1}$ for $k \geq 1$. An example word from Z_3 is 00100021003000100200 10. Note that every suffix of $x \in Z_k$ that starts with 0 belongs to L_k and every suffix of $x \in Z_k$ that starts with $a > 0$ does not belong to L_k. The former follows by induction on k; the latter holds since $L_k \subseteq 0\,\mathbb{N}^*$.

Fix some $k \geq 1$ and let $\mathcal{B} = (\mathcal{B}_n)_{n \geq 0}$ be a fixed-size sliding window algorithm for L_k. Consider a window size n. We claim that $\mathcal{B}_n(x) \neq \mathcal{B}_n(y)$ for all $x, y \in Z_k$ with $|x| = |y| = n$ and $x \neq y$. To see this, write $x = zau$ and $y = zbv$ with $a, b \in \{0, \ldots, k\}$, $a \neq b$. We must have $a = 0$ and $b > 0$ or vice versa. Assume that $a = 0$ and $b > 0$. Thus, $au \in L_k$ and $bv \notin L_k$. Hence, we have $\mathrm{wnd}(x0^{|z|}) = au0^{|z|} \in L_k$ and $\mathrm{wnd}(y0^{|z|}) = bv0^{|z|} \notin L_k$. But if $\mathcal{B}_n(x) = \mathcal{B}_n(y)$, then also $\mathcal{B}_n(x0^{|z|}) = \mathcal{B}_n(y0^{|z|})$, which yields a contradiction.

It follows that $\mathcal{B}_n$ has at least $\binom{n}{2^k-1} \geq (n/(2^k-1))^{2^k-1} \geq (n/2^k)^{2^k-1}$ many states, which implies $v_{\mathcal{B}}(n) \geq (2^k - 1) \cdot (\log n - k)$.

6 Constant space sliding-window algorithms

Lemma 4.1 implies that $V_L(n) \geq \log n$ if $\emptyset \neq L \neq \Sigma^*$. Thus, only trivial languages have a constant-space variable-size streaming algorithm. This changes in the fixed-size window model. In [20] we characterized those regular languages L in $\mathsf{F}(\mathcal{O}(1))$ in terms of the left Cayley graph of the syntactic monoid of L. Here we give a more natural characterization.

A language $L \subseteq \Sigma^*$ is called *k-suffix testable* if for all $x, y \in \Sigma^*$ and $z \in \Sigma^k$ we have: $xz \in L \iff yz \in L$. Equivalently, L is a Boolean combination of languages of the form $\Sigma^* w$ where $w \in \Sigma^{\leq k}$. We call L *suffix testable* if it is k-suffix testable for some $k \geq 0$. Clearly, every finite language is suffix testable: if $L \subseteq \Sigma^{\leq k}$ then L is $(k+1)$-suffix testable. The class of suffix testable languages corresponds to the variety $\mathbf{D}$ of definite monoids [32]. Our main result about the class $\mathsf{F}(\mathcal{O}(1))$ is the following; its proof can be found in the full version [19]:

▶ **Theorem 6.1.** *A regular language $L \subseteq \Sigma^*$ belongs to $\mathsf{F}(\mathcal{O}(1))$ if and only if L is a finite Boolean combination of suffix testable languages and regular length languages.*

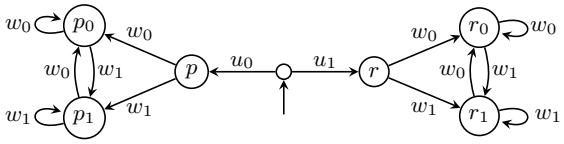


Figure 2 A critical tuple (u_0, u_1, w_0, w_1).

7 Deciding space complexity in the sliding window model

In this section, we consider the complexity of the following decision problems:

- DFA(1): Given a DFA $\mathcal{A}$, does $L(\mathcal{A})$ belong to $\mathsf{F}(\mathcal{O}(1))$?
- NFA(1): Given an NFA $\mathcal{A}$, does $L(\mathcal{A})$ belong to $\mathsf{F}(\mathcal{O}(1))$?
- DFA($\log n$): Given a DFA $\mathcal{A}$, does $L(\mathcal{A})$ belong to $\mathsf{F}(\mathcal{O}(\log n)) = \mathsf{V}(\mathcal{O}(\log n))$?
- NFA($\log n$): Given an NFA $\mathcal{A}$, does $L(\mathcal{A})$ belong to $\mathsf{F}(\mathcal{O}(\log n)) = \mathsf{V}(\mathcal{O}(\log n))$?

Recall that by Theorem 3.3, $L(\mathcal{A})$ belongs to $\mathsf{V}(\mathcal{O}(1))$ iff $L(\mathcal{A})$ is trivial. The latter problem can be shown to be NL-complete (resp., PSPACE-complete) if $\mathcal{A}$ is a DFA (resp., an NFA) using standard constructions. The same complexity bounds hold for the above problems:

▶ **Theorem 7.1.** *The following hold:*

- DFA(1) *and* DFA($\log n$) *are* NL*-complete.*
- NFA(1) *and* NFA($\log n$) *are* PSPACE*-complete.*

We only sketch the NL upper bound for DFA($\log n$); the other parts of Theorem 7.1 are shown in the full version [19]. We can assume that the input DFA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ is minimal; see the argument for the NL upper bound for DFA(1) in [19]. Let $L = L(\mathcal{A})$. For $u, x_0, x_1 \in \Sigma^*$ we define $Q(u, x_0, x_1) = \{\mathcal{A}(ux) : x \in \{x_0, x_1\}^*\}$, which is the set of states of $\mathcal{A}$ reachable from the initial state by first reading u and then an arbitrary product of copies of x_0 and x_1. We call a tuple (u_0, u_1, w_0, w_1) of words *critical*, if (i) $|u_0| = |u_1| \geq 1$, (ii) u_i is a suffix of w_i for all $i \in \{0, 1\}$ and (iii) $Q(u_0, w_0, w_1) \cap Q(u_1, w_0, w_1) = \emptyset$. Using critical tuples, we can state another characterization of the class $\mathsf{V}_{\mathsf{reg}}(\log n)$:

▶ **Lemma 7.2.** *We have $L \notin \mathsf{V}_{\mathsf{reg}}(\log n)$ if and only if there exists a critical tuple in $\mathcal{A}$.*

We show that if there exists a critical tuple, then there exists a critical tuple (u_0, u_1, w_0, w_1) such that $|Q(u_0, w_0, w_1)| \leq 3 \geq |Q(u_1, w_0, w_1)|$. Assume that (u_0, u_1, w_0, w_1) is a critical tuple. Let $h\colon \Sigma^* \to M$ be the canonical homomorphism into the transition monoid M of $\mathcal{A}$, which right acts on Q via $Q \times M \to Q$, $(q, m) \mapsto q \cdot m = m(q)$. Notice that $Q(u_i, w_0, w_1) = \{\mathcal{A}(u_i) \cdot m : m \in \{h(w_0), h(w_1)\}^*\}$, where X^* denotes the submonoid of M generated by $X \subseteq M$. It suffices to define a new critical tuple (u_0, u_1, x_0, x_1) with the property that $h(x_i) \cdot h(x_j) = h(x_j)$ for all $i, j \in \{0, 1\}$. This implies $\{h(x_0), h(x_1)\}^* = \{1, h(x_0), h(x_1)\}$, and hence, $|Q(u_0, x_0, x_1)| \leq 3 \geq |Q(u_1, x_0, x_1)|$.

Notice that if (u_0, u_1, w_0, w_1) is critical, then also $(u_0, u_1, y_0 w_0, y_1 w_1)$ is critical for all $y_0, y_1 \in \{w_0, w_1\}^*$. Let $\omega \geq 1$ be a number such that m^ω is idempotent for all $m \in M$. By choosing $e_0 = (h(w_0)^\omega h(w_1)^\omega)^\omega h(w_0)^\omega$ and $e_1 = (h(w_0)^\omega h(w_1)^\omega)^\omega$ we indeed obtain $e_i e_j = e_j$ for all $i, j \in \{0, 1\}$. Hence we define $x_0 = (w_0^\omega w_1^\omega)^\omega w_0^\omega$ and $x_1 = (w_0^\omega w_1^\omega)^\omega$.

To decide whether $L \notin \mathsf{V}_{\mathsf{reg}}(\log n)$ it therefore suffices to check whether there is a critical tuple (u_0, u_1, w_0, w_1) such that $|Q(u_0, w_0, w_1)| \leq 3 \geq |Q(u_1, w_0, w_1)|$. Figure 2 illustrates the substructure we need to detect in $\mathcal{A}$. We show that the existence of such a structure can be verified in NL. To do so, we reduce to testing emptiness of one-counter automata, which is known to be in NL [25]. For two states $p, r \in Q$ let $\mathcal{A}_{p,r} = (Q, \Sigma, p, \delta, \{r\})$ be the automaton

$\mathcal{A}$ with initial state p and final state r, and let $L(p,r) = L(\mathcal{A}_{p,r})$. The algorithm iterates over all disjoint sets $\{p, p_0, p_1\}, \{r, r_0, r_1\} \subseteq Q$. For $i \in \{0,1\}$ let $\mathcal{A}_i$ be a DFA for the language $L(p,p_i) \cap L(p_0,p_i) \cap L(p_1,p_i) \cap L(r,r_i) \cap L(r_0,r_i) \cap L(r_1,r_i)$. Now consider the language $\{v_0 \# u_0 \# v_1 \# u_1 : v_i u_i \in L(\mathcal{A}_i) \text{ for } i \in \{0,1\}, |u_0| = |u_1| \geq 1, u_0 \in L(q_0,p), u_1 \in L(q_0,r)\}$ for which one can construct in logspace a one-counter automaton. The counter is used to verify the constraint $|u_0| = |u_1|$. The language above is empty if and only if $\mathcal{A}$ has a critical tuple. This concludes the proof that $\textsc{Dfa}(\log n)$ belongs to NL.

References

1 Charu C. Aggarwal. *Data Streams - Models and Algorithms*. Springer, 2007.

2 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

3 Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of PODS 2004*, pages 286–296. ACM, 2004.

4 Brian Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of PODS 2003*, pages 234–243. ACM, 2003.

5 Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theoretical Computer Science*, 494:13–23, 2013.

6 Ajesh Babu, Nutan Limaye, and Girish Varma. Streaming algorithms for some problems in log-space. In *Proceedings of TAMC 2010*, volume 6108 of *Lecture Notes in Computer Science*, pages 94–104. Springer, 2010.

7 Vladimir Braverman. Sliding window algorithms. In *Encyclopedia of Algorithms*, pages 2006–2011. Springer, 2016.

8 Vladimir Braverman and Rafail Ostrovsky. Smooth histograms for sliding windows. In *Proceedings of FOCS 2007*, pages 283–293. IEEE Computer Society, 2007.

9 Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. Optimal sampling from sliding windows. *J. Comput. Syst. Sci.*, 78(1):260–272, 2012.

10 Dany Breslauer and Zvi Galil. Real-time streaming string-matching. *ACM Trans. Algorithms*, 10(4):22:1–22:12, 2014.

11 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. Dictionary matching in a stream. In *Proceedings of ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 361–372. Springer, 2015.

12 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. The k-mismatch problem revisited. In *Proceedings of SODA 2016*, pages 2039–2052. SIAM, 2016.

13 Raphaël Clifford and Tatiana A. Starikovskaya. Approximate hamming distance in a stream. In *Proceedings of ICALP 2016*, volume 55 of *LIPIcs*, pages 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

14 Michael S. Crouch, Andrew McGregor, and Daniel Stubbs. Dynamic graphs in the sliding-window model. In *Proceedings of ESA 2013*, volume 8125 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2013.

15 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002.

16 A. Policriti F. Parlamento and K. Rao. Witnessing differences without redundancies. *Proceedings of the American Mathematical Society*, 125(2):587–594, 1997.

17 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

18 Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre. Streaming property testing of visibly pushdown languages. In *Proceedings of ESA 2016*, volume 57 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

19 Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. Technical report, arXiv.org, 2018. `https://arxiv.org/abs/1702.04376`.

20 Moses Ganardi, Danny Hucke, and Markus Lohrey. Querying regular languages over sliding windows. In *Proceedings of FSTTCS 2016*, volume 65 of *LIPIcs*, pages 18:1–18:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

21 Pawel Gawrychowski, Dalia Krieger, Narad Rampersad, and Jeffrey Shallit. Finding the growth rate of a regular or context-free language in polynomial time. *International Journal on Foundations of Computer Science*, 21(4):597–618, 2010.

22 Lukasz Golab and M. Tamer Özsu. Processing sliding window multi-joins in continuous queries over data streams. In *Proceedings of VLDB 2003*, pages 500–511. Morgan Kaufmann, 2003.

23 Christian Konrad and Frédéric Magniez. Validating XML documents in the streaming model with external memory. *ACM Trans. Database Syst.*, 38(4):27:1–27:36, 2013.

24 Andreas Krebs, Nutan Limaye, and Srikanth Srinivasan. Streaming algorithms for recognizing nearly well-parenthesized expressions. In *Proceedings of MFCS 2011*, volume 6907 of *Lecture Notes in Computer Science*, pages 412–423. Springer, 2011.

25 Michel Latteux. Langages á un compteur. *Journal of Computer and System Sciences*, 26(1):14–33, 1983.

26 Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. *SIAM J. Comput.*, 43(6):1880–1905, 2014.

27 Philip M. Lewis II, Richard Edwin Stearns, and Juris Hartmanis. Memory bounds for recognition of context-free and context-sensitive languages. In *Proceedings of SWCT (FOCS) 1965*, pages 191–202. IEEE Computer Society, 1965.

28 J. Ian Munro and Mike Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980.

29 Luc Segoufin and Cristina Sirangelo. Constant-memory validation of streaming XML documents against dtds. In *Proceedings of ICDT 2007*, volume 4353 of *Lecture Notes in Computer Science*, pages 299–313. Springer, 2007.

30 Luc Segoufin and Victor Vianu. Validating streaming XML documents. In *Proceedings of PODS 2002*, pages 53–64. ACM, 2002.

31 Richard Edwin Stearns, Juris Hartmanis, and Philip M. Lewis II. Hierarchies of memory limited computations. In *Proceedings of SWCT (FOCS) 1965*, pages 179–190. IEEE Computer Society, 1965.

32 Howard Straubing. Finite semigroup varieties of the form $V * D$. *Journal of Pure and Applied Algebra*, 36:53–94, 1985.

33 Andrew Szilard, Sheng Yu, Kaizhong Zhang, and Jeffrey Shallit. Characterizing regular languages with polynomial densities. In *Proceedings of MFCS 1992*, volume 629 of *Lecture Notes in Computer Science*, pages 494–503. Springer, 1992.

Knapsack Problems for Wreath Products

Moses Ganardi
Universität Siegen, Germany
ganardi@eti.uni-siegen.de

Daniel König
Universität Siegen, Germany
koenig@eti.uni-siegen.de

Markus Lohrey
Universität Siegen, Germany
lohrey@eti.uni-siegen.de

Georg Zetzsche
LSV, CNRS & ENS Paris-Saclay, France
zetzsche@lsv.fr

Abstract

In recent years, knapsack problems for (in general non-commutative) groups have attracted attention. In this paper, the knapsack problem for wreath products is studied. It turns out that decidability of knapsack is not preserved under wreath product. On the other hand, the class of knapsack-semilinear groups, where solutions sets of knapsack equations are effectively semilinear, is closed under wreath product. As a consequence, we obtain the decidability of knapsack for free solvable groups. Finally, it is shown that for every non-trivial abelian group G, knapsack (as well as the related subset sum problem) for the wreath product $G \wr \mathbb{Z}$ is NP-complete.

2012 ACM Subject Classification Computing methodologies → Equation and inequality solving algorithms

Keywords and phrases Knapsack, Wreath Products, Decision Problems in Group Theory

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.32

Related Version A full version of the paper is available at `http://arxiv.org/abs/1709.09598`, [4].

Funding The fourth author is supported by a fellowship within the Postdoc-Program of the German Academic Exchange Service (DAAD) and by Labex DigiCosme, Univ. Paris-Saclay, project VERICONISS. The second and third author are supported by the DFG grant LO 748/12-1.

1 Introduction

In [15], Myasnikov, Nikolaev, and Ushakov began the investigation of classical discrete optimization problems, which are formulated over the integers, for arbitrary (possibly non-commutative) groups. The general goal of this line of research is to study to what extent results from the commutative setting can be transferred to the non-commutative setting. Among other problems, Myasnikov et al. introduced for a finitely generated group G the *knapsack problem* and the *subset sum problem*. The input for the knapsack problem is a sequence of group elements $g_1, \ldots, g_k, g \in G$ (specified by finite words over the generators of G) and it is asked whether there exists a solution $(x_1, \ldots, x_k) \in \mathbb{N}^k$ of the equation $g_1^{x_1} \cdots g_k^{x_k} = g$. For the subset sum problem one restricts the solution to $\{0,1\}^k$. For the

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 32; pp. 32:1–32:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

particular case $G = \mathbb{Z}$ (where the additive notation $x_1 \cdot g_1 + \cdots + x_k \cdot g_k = g$ is usually preferred) these problems are NP-complete (resp., TC^0-complete) if the numbers $g_1, \ldots, g_k, g$ are encoded in binary representation [7, 6] (resp., unary notation [2]).

Another motivation is that decidability of knapsack for a group G implies that the membership problem for every fixed polycyclic subgroup of G is decidable. This follows from the well-known fact that every polycyclic group A has a generating set $\{a_1, \ldots, a_k\}$ such that every element of A can be written as $a_1^{n_1} \cdots a_k^{n_k}$ for $n_1, \ldots, n_k \in \mathbb{N}$, see e.g. [17, Chapter 9].

In [15], Myasnikov et al. encode elements of the finitely generated group G by words over the group generators and their inverses, which corresponds to the unary encoding of integers. There is also an encoding of words that corresponds to the binary encoding of integers, so called straight-line programs, and knapsack problems under this encodings have been studied in [11]. In this paper, we only consider the case where input words are explicitly represented. Here is a (non-complete) list of known results concerning knapsack and subset sum problems:

- Subset sum and knapsack can be solved in polynomial time for every hyperbolic group [15]. In [3] this result was extended to free products of any number of hyperbolic groups and finitely generated abelian groups.
- For every virtually nilpotent group, subset sum belongs to nondeterministic logspace [8]. On the other hand, there are nilpotent groups of class 2 for which knapsack is undecidable. Examples are direct products of sufficiently many copies of the discrete Heisenberg group $H_3(\mathbb{Z})$ [8], and free nilpotent groups of class 2 and sufficiently high rank [14].
- Knapsack for $H_3(\mathbb{Z})$ is decidable [8]. In particular, together with the previous point it follows that decidability of knapsack is not preserved under direct products.
- For the following groups, subset sum is NP-complete (whereas the word problem can be solved in polynomial time): free metabelian non-abelian groups of finite rank, the wreath product $\mathbb{Z} \wr \mathbb{Z}$, Thompson's group F, the Baumslag-Solitar group $\mathrm{BS}(1, 2)$ [15], and every polycyclic group that is not virtually nilpotent [16].
- Knapsack is decidable for every co-context-free group [8].
- Knapsack belongs to NP for all virtually special groups (finite extensions of subgroups of graph groups) [11]. For graph groups (also known as right-angled Artin groups) a complete classification of the complexity of knapsack was obtained in [12]: If the underlying graph contains an induced path or cycle on 4 nodes, then knapsack is NP-complete; in all other cases knapsack can be solved in polynomial time (even in LogCFL).
- Decidability of knapsack is preserved under finite extensions, HNN-extensions over finite associated subgroups and amalgamated free products over finite subgroups [11].

In this paper, we study the knapsack problem for wreath products. The wreath product is a fundamental construction in group theory and semigroup theory, see Section 4 for the definition. An important application of wreath products in group theory is the Magnus embedding theorem [18], which allows to embed the quotient group $F_k/[N, N]$ into the wreath product $\mathbb{Z}^k \wr (F_k/N)$, where F_k is a free group of rank k and N is a normal subgroup of F_k. In particular, free solvable groups can be embedded into iterated wreath products of free abelian groups; a fact that we will use in this paper. Wreath products also have some nice algorithmic properties: The word problem for a wreath product $G \wr H$ is AC^0-reducible to the word problems for the factors G and H, and the conjugacy problem for $G \wr H$ is TC^0-reducible to the conjugacy problems for G and H and the so called power problem for H [13].

As in the case of direct products, it turns out that decidability of knapsack is not preserved under wreath products: For this we consider direct products of the form $H_3(\mathbb{Z}) \times \mathbb{Z}^\ell$, where $H_3(\mathbb{Z})$ is the discrete 3-dimensional Heisenberg group. It was shown in [8] that for every $\ell \geq 0$, knapsack is decidable for $H_3(\mathbb{Z}) \times \mathbb{Z}^\ell$. We prove that for every non-trivial group G and every sufficiently large ℓ, knapsack for $G \wr (H_3(\mathbb{Z}) \times \mathbb{Z}^\ell)$ is undecidable.

By the above discussion, we need stronger assumptions on G and H to obtain decidability of knapsack for $G \wr H$. We exhibit a very weak condition on G and H, knapsack-semilinearity, which is sufficient for decidability of knapsack for $G \wr H$. A finitely generated group G is knapsack-semilinear if for every knapsack equation, the set of all solutions (a solution can be seen as an vector of natural numbers) is effectively semilinear.

Clearly, for every knapsack-semilinear group, the knapsack problem is decidable. While the converse is not true, the class of knapsack-semilinear groups is extraordinarily wide. The simplest examples are finitely generated abelian groups, but it also includes the rich class of virtually special groups [11], all hyperbolic groups [4], and all co-context-free groups [8]. Furthermore, it is known to be closed under direct products (an easy observation), going to a finitely generated subgroup, going to a finite extension, HNN-extensions over finite associated subgroups and amalgamated free products over finite subgroups (the last three closure properties are simple extensions of the transfer theorems in [11]). In fact, the only non-knapsack-semilinear groups with a decidable knapsack problem that we are aware of are the groups $H_3(\mathbb{Z}) \times \mathbb{Z}^n$ for $n \geq 0$.

We prove in Section 6 that the class of knapsack-semilinear groups is closed under wreath products. As a direct consequence of the Magnus embedding, it follows that knapsack is decidable for every free solvable group. Recall that, in contrast, knapsack for free nilpotent groups is in general undecidable [14].

Finally, we consider the complexity of knapsack for wreath products. We prove that for every non-trivial finitely generated abelian group G, knapsack for $G \wr \mathbb{Z}$ is NP-complete (the hard part is membership in NP). This result includes important special cases like for instance the lamplighter group $\mathbb{Z}_2 \wr \mathbb{Z}$ and $\mathbb{Z} \wr \mathbb{Z}$. Wreath products of the form $G \wr \mathbb{Z}$ with G abelian turn out to be important in connection with subgroup distortion [1]. Our proof also shows that for every non-trivial finitely generated abelian group G, the subset sum problem for $G \wr \mathbb{Z}$ is NP-complete. In [15] this result is only shown for infinite abelian groups G.

Missing proofs can be found in the full version [4].

2 Preliminaries

We assume standard notions concerning groups. A group G is *finitely generated* if there exists a finite subset $\Sigma \subseteq G$ such that every element $g \in G$ can be written as $g = a_1 a_2 \cdots a_n$ with $a_1, a_2, \ldots, a_n \in \Sigma$. We also say that the word $a_1 a_2 \cdots a_n \in \Sigma^*$ evaluates to g (or represents g). The set Σ is called a finite generating set of G. We always assume that Σ is symmetric in the sense that $a \in \Sigma$ implies $a^{-1} \in \Sigma$. Elements of G will be represented by words from Σ^*. An element $g \in G$ is called *torsion element* if there is an $n \geq 1$ with $g^n = 1$. The smallest such n is the *order* of g and is denoted by $\mathrm{ord}(g)$. If g is not a torsion element, we set $\mathrm{ord}(g) = \infty$.

A set $A \subseteq \mathbb{N}^k$ is *linear* if $A = \{v_0 + \lambda_1 \cdot v_1 + \cdots + \lambda_n \cdot v_n \mid \lambda_1, \ldots, \lambda_n \in \mathbb{N}\}$ for vectors $v_0, \ldots, v_n \in \mathbb{N}^k$. The tuple of vectors $(v_0, \ldots, v_n)$ is a *linear representation* of A. A set $A \subseteq \mathbb{N}^k$ is *semilinear* if it is a finite union of linear sets $A_1, \ldots, A_m$. A *semilinear representation* of A is a list of linear representations for the linear sets $A_1, \ldots, A_m$. It is well-known that the semilinear subsets of $\mathbb{N}^k$ are exactly the sets definable in *Presburger arithmetic*. These are those sets that can be defined with a first-order formula $\varphi(x_1, \ldots, x_k)$ over the structure $(\mathbb{N}, 0, +, \leq)$ [5]. Moreover, the transformations between such a first-order formula and an equivalent semilinear representation are effective. In particular, the semilinear sets are effectively closed under Boolean operations.

3 Knapsack for groups

Let G be a finitely generated group with the finite symmetric generating set Σ. Moreover, let V be a set of formal variables that take values from $\mathbb{N}$. For a subset $U \subseteq V$, we use $\mathbb{N}^U$ to denote the set of maps $\nu\colon U \to \mathbb{N}$, which we call *valuations*. An *exponent expression* over G is a formal expression of the form $E = v_0 u_1^{x_1} v_1 u_2^{x_2} v_2 \cdots u_k^{x_k} v_k$ with $k \geq 0$ and words $u_i, v_i \in \Sigma^*$. Here, the variables do not have to be pairwise distinct. If every variable in an exponent expression occurs at most once, it is called a *knapsack expression*. Let $V_E = \{x_1, \ldots, x_k\}$ be the set of variables that occur in E. For a valuation $\nu \in \mathbb{N}^U$ such that $V_E \subseteq U$ (in which case we also say that ν is a valuation for E), we define $\nu(E) = v_0 u_1^{\nu(x_1)} v_1 u_2^{\nu(x_2)} v_2 \cdots u_k^{\nu(x_k)} v_k \in \Sigma^*$. We say that ν is a *solution* of the equation $E = 1$ if $\nu(E)$ evaluates to the identity element 1 of G. With $\mathsf{Sol}(E)$ we denote the set of all solutions $\nu \in \mathbb{N}^{V_E}$ of E. We can view $\mathsf{Sol}(E)$ as a subset of $\mathbb{N}^k$. The *length* of E is defined as $|E| = |v_0| + \sum_{i=1}^{k} |u_i| + |v_i|$, whereas k is its *depth*. If the length of a knapsack expression is not needed, we will write an exponent expression over G also as $E = h_0 g_1^{x_1} h_1 g_2^{x_2} h_2 \cdots g_k^{x_k} h_k$ where $g_i, h_i \in G$. We define *solvability of exponent equations over* G, $\textsc{ExpEq}(G)$ for short, as the following decision problem:

Input: A finite list of exponent expressions $E_1, \ldots, E_n$ over G.
Question: Is $\bigcap_{i=1}^{n} \mathsf{Sol}(E_i)$ non-empty?

The knapsack problem for G, $\mathrm{KP}(G)$ for short, is the following decision problem:

Input: A single knapsack expression E over G.
Question: Is $\mathsf{Sol}(E)$ non-empty?

We also consider the uniform knapsack problem for powers $G^m = \prod_{i=1}^{m} G_i$ with $G_i \cong G$. We denote this problem with $\mathrm{KP}(G^*)$. Formally, it is defined as follows:

Input: A number $m \geq 0$ (in unary notation) and a knapsack expression E over G^m.
Question: Is $\mathsf{Sol}(E)$ non-empty?

It turns out that the problems $\mathrm{KP}(G^*)$ and $\textsc{ExpEq}(G)$ are inter-reducible:

▶ **Proposition 3.1.** *$\mathrm{KP}(G^*)$ is decidable if and only if $\textsc{ExpEq}(G)$ is decidable.*

Note that the exponent equation $v_0 u_1^{x_1} v_1 u_2^{x_2} v_2 \cdots u_k^{x_k} v_k = 1$ is equivalent to the exponent equation $(v_0 u_1 v_0^{-1})^{x_1} (v_0 v_1 u_2 v_1^{-1} v_0^{-1})^{x_2} \cdots (v_0 \cdots v_{k-1} u_k v_{k-1}^{-1} \cdots v_0^{-1})^{x_k} (v_0 \cdots v_k) = 1$. Hence, it suffices to consider exponent expressions of the form $u_1^{x_1} u_2^{x_2} \cdots u_k^{x_k} v$.

The group G is called *knapsack-semilinear* if for every knapsack expression E over G, the set $\mathsf{Sol}(E)$ is a semilinear set of vectors and a semilinear representation can be effectively computed from E. The following classes of groups only contain knapsack-semilinear groups:

- virtually special groups [11]: these are finite extensions of subgroups of graph groups (aka right-angled Artin groups). The class of virtually special groups is very rich. It contains all Coxeter groups, one-relator groups with torsion, fully residually free groups, and fundamental groups of hyperbolic 3-manifolds.
- hyperbolic groups [4]
- co-context-free groups [8], i.e., groups where the set of all words over the generators that do not represent the identity is a context-free language. Lehnert and Schweitzer [9] have shown that the Higman-Thompson groups are co-context-free.

Since emptiness of the intersection of finitely many semilinear sets is decidable, we have:

▶ **Lemma 3.2.** *If G is knapsack-semilinear, then $\mathrm{KP}(G^*)$ and $\textsc{ExpEq}(G)$ are decidable.*

An example of a group G, where $\mathrm{KP}(G)$ is decidable, but $\mathrm{KP}(G^*)$ and $\textsc{ExpEq}(G)$ are undecidable is the Heisenberg group $H_3(\mathbb{Z})$ (the group of all upper triangular (3×3)-matrices over $\mathbb{Z}$ with all diagonal entries equal to 1) [8]. Hence, $H_3(\mathbb{Z})$ is not knapsack-semilinear.

4 Wreath products

Let G and H be groups. Consider the direct sum $K = \bigoplus_{h \in H} G_h$, where G_h is a copy of G. We view K as the set $G^{(H)}$ of all mappings $f\colon H \to G$ such that $\mathsf{supp}(f) = \{h \in H \mid f(h) \neq 1\}$ is finite, together with pointwise multiplication as the group operation. The set $\mathsf{supp}(f) \subseteq H$ is called the *support* of f. The group H has a natural left action on $G^{(H)}$ given by $hf(a) = f(h^{-1}a)$, where $f \in G^{(H)}$ and $h, a \in H$. The corresponding semidirect product $G^{(H)} \rtimes H$ is the *wreath product* $G \wr H$. In other words:

- Elements of $G \wr H$ are pairs (f, h), where $h \in H$ and $f \in G^{(H)}$.
- The multiplication in $G \wr H$ is defined as follows: Let $(f_1, h_1), (f_2, h_2) \in G \wr H$. Then $(f_1, h_1)(f_2, h_2) = (f, h_1h_2)$, where $f(a) = f_1(a)f_2(h_1^{-1}a)$.

The following intuition might be helpful: An element $(f, h) \in G \wr H$ can be thought of as a finite multiset of elements of $G \setminus \{1_G\}$ that are sitting at certain elements of H (the map f) together with the distinguished element $h \in H$, which can be thought of as a cursor moving in H. The product $(f_1, h_1)(f_2, h_2)$ is computed as follows. First, we shift the finite collection of G-elements (that corresponds to the mapping f_2) by h_1: If $g \in G \setminus \{1_G\}$ is sitting at $a \in H$ (i.e., $f_2(a) = g$), then we remove g from a and put it to the new location $h_1a \in H$. This new collection corresponds to the mapping $f_2'\colon a \mapsto f_2(h_1^{-1}a)$. After this shift, the two collections of G-elements are multiplied pointwise: If in $a \in H$ the elements g_1 and g_2 are sitting (i.e., $f_1(a) = g_1$ and $f_2'(a) = g_2$), then we put the product g_1g_2 into the location a. Finally, the new distinguished H-element (the new cursor position) becomes h_1h_2.

By identifying $f \in G^{(H)}$ with $(f, 1_H) \in G \wr H$ and $h \in H$ with $(1_{G^{(H)}}, h)$, we regard $G^{(H)}$ and H as subgroups of $G \wr H$. Hence, for $f \in G^{(H)}$ and $h \in H$, we have $fh = (f, 1_H)(1_{G^{(H)}}, h) = (f, h)$. There are two natural projection maps $\sigma_{G \wr H}\colon G \wr H \to H$ (which is a morphism) and $\tau_{G \wr H}\colon G \wr H \to G^{(H)}$ with $\sigma_{G \wr H}(f, h) = h$ and $\tau_{G \wr H}(f, h) = f$. If G (resp. H) is generated by the set Σ (resp. Γ) with $\Sigma \cap \Gamma = \emptyset$, then $G \wr H$ is generated by the set $\{(f_a, 1_H) \mid a \in \Sigma\} \cup \{(f_{1_G}, b) \mid b \in \Gamma\}$, where for $g \in G$, the mapping $f_g : H \to G$ is defined by $f_g(1_H) = g$ and $f_g(x) = 1_G$ for $x \in H \setminus \{1_H\}$. We identify this generating set with $\Sigma \uplus \Gamma$.

5 Main results

In this section, we state the main results of the paper. We begin with a general necessary condition for knapsack to be decidable for a wreath product. Note that if H is finite, then $G \wr H$ is a finite extension of $G^{|H|}$ [10, Proposition 1], meaning that $\mathrm{KP}(G \wr H)$ is decidable if and only if $\mathrm{KP}(G^{|H|})$ is decidable [11, Theorem 11] (in [11], preservation of NP-membership was shown, but the proof also yields preservation of decidability). Therefore, we are only interested in the case that H is infinite.

▶ **Proposition 5.1.** *Suppose H is infinite. If $\mathrm{KP}(G \wr H)$ is decidable, then $\mathrm{KP}(H)$ and $\mathrm{KP}(G^*)$ are decidable.*

Of course, as a subgroup of $G \wr H$, H inherits decidability of knapsack. On the other hand, given $m \in \mathbb{N}$, one can easily compute an embedding of G^m into $G \wr H$ and thus solve knapsack instances over G^m uniformly in m. Proposition 5.1 shows that $\mathrm{KP}(H_3(\mathbb{Z}) \wr \mathbb{Z})$ is undecidable: It was shown in [8] that $\mathrm{KP}(H_3(\mathbb{Z}))$ is decidable, whereas for some $m > 1$, the problem $\mathrm{KP}(H_3(\mathbb{Z})^m)$ is undecidable.

Proposition 5.1 raises the question whether decidability of $\mathrm{KP}(H)$ and $\mathrm{KP}(G^*)$ implies decidability of $\mathrm{KP}(G \wr H)$. We disprove this in the following theorem. The second statement is due to the fact that for every $\ell \in \mathbb{N}$, $\mathrm{KP}(H_3(\mathbb{Z}) \times \mathbb{Z}^\ell)$ is decidable, as shown in [8].

▶ **Theorem 5.2.** *There is an $\ell \in \mathbb{N}$ such that for every $G \neq 1$, $\mathrm{KP}(G \wr (H_3(\mathbb{Z}) \times \mathbb{Z}^\ell))$ is undecidable. In particular, there are groups G, H such that $\mathrm{KP}(G^*)$ and $\mathrm{KP}(H)$ are decidable and $\mathrm{KP}(G \wr H)$ is undecidable.*

We therefore need to strengthen the assumptions on H in order to show decidability of $\mathrm{KP}(G \wr H)$. Under the weak additional assumption of knapsack-semilinearity for H, we obtain a partial converse to Proposition 5.1. In Section 6 we prove:

▶ **Theorem 5.3.** *Let H be knapsack-semilinear and infinite. Then $\mathrm{KP}(G \wr H)$ is decidable if and only if $\mathrm{KP}(G^*)$ is decidable.*

If G is knapsack-semilinear, the solution sets are effectively semilinear:

▶ **Theorem 5.4.** *The group $G \wr H$ is knapsack-semilinear if and only if both G and H are knapsack-semilinear.*

Since every free abelian group is clearly knapsack-semilinear, it follows that the iterated wreath products $G_{1,r} = \mathbb{Z}^r$ and $G_{d+1,r} = \mathbb{Z}^r \wr G_{d,r}$ are knapsack-semilinear. By the well-known Magnus embedding, the free solvable group $S_{d,r}$ embeds into $G_{d,r}$. Hence, we get:

▶ **Corollary 5.5.** *Every free solvable group is knapsack-semilinear. Hence, solvability of exponent equations is decidable for free solvable groups.*

Finally, we consider the complexity of knapsack for wreath products. In Section 7 we prove NP-completeness for an important special case:

▶ **Theorem 5.6.** *For every finitely generated abelian group $G \neq 1$, $\mathrm{KP}(G \wr \mathbb{Z})$ is NP-complete.*

6 (Un)decidability: Proofs of Theorems 5.2, 5.3, and 5.4

Undecidability. We begin with a proof sketch for Theorem 5.2. Here, the only property of $H_3(\mathbb{Z})$ that we use is that solvability of knapsack instances of some fixed depth k over the group $H_3(\mathbb{Z})^m$ is undecidable for some $m \geq 0$, which was shown in [8]. Using this property, we prove that $\mathrm{KP}(G^m \wr (H_3(\mathbb{Z}) \times \mathbb{Z}^{k \cdot m}))$ is undecidable. Since every group $G^n \wr H$ embeds into $G \wr (H \times \mathbb{Z})$, this implies undecidability of $\mathrm{KP}(G \wr (H_3(\mathbb{Z}) \times \mathbb{Z}^{k \cdot m+1}))$.

Undecidability of $\mathrm{KP}(G^m \wr (H_3(\mathbb{Z}) \times \mathbb{Z}^{k \cdot m}))$ is shown as follows. Consider a knapsack expression E over $H_3(\mathbb{Z})^m$ of depth k. We turn E into m knapsack expressions $E_1, \ldots, E_m$ over $H_3(\mathbb{Z})$ of depth k so that E has a solution if and only if there is a common solution to $E_1, \ldots, E_m$ so that for every $i \in [1, k]$, the i-th variable for each expression has the same value. We construct a knapsack expression over $G^m \wr (H_3(\mathbb{Z}) \times \mathbb{Z}^{k \cdot m})$ as follows. We pick an $a \in G \setminus \{1\}$ and use it as a "breadcrumb": It is placed at a particular cursor position in $H_3(\mathbb{Z}) \times \mathbb{Z}^{k \cdot m}$ in one of the m coordinates of G^m and is later collected by multiplying a^{-1} in the same coordinate of G^m. Fix a correspondence between the $k \cdot m$ variables in $E_1, \ldots, E_m$ and the coordinates of $\mathbb{Z}^{k \cdot m}$. Our new expression operates in three phases. The first phase performs for each $j = 1, \ldots, m$ the following. It places a breadcrumb in the j-th coordinate of G^m and then moves the cursor by some value of E_j. At the same time, for each variable in E_j, it moves the cursor in the corresponding coordinate of $\mathbb{Z}^{k \cdot m}$ by the value of that variable.

In the second phase, we check that for each $i \in [1, k]$, the i-th variable for each expression has the same value and move the cursor back to the origin. To this end, we move the cursor in the $\mathbb{Z}^{k \cdot m}$-coordinates so that two coordinates that correspond to two variables that are to be compared are decremented simultaneously. After this, we collect the first breadcrumb. In the third phase, it remains to check that each E_j evaluates to $1 \in H_3(\mathbb{Z})$. Since we are already in the origin, this amounts to checking that we can collect the remaining breadcrumbs $2, \ldots, m$ by moving just in the $\mathbb{Z}^{k \cdot m}$-coordinates. The full proof can be found in [4].

Decidability. The rest of this section is devoted to the positive results, Theorems 5.3 and 5.4. Let us fix a wreath product $G \wr H$. Recall the projections $\sigma = \sigma_{G \wr H} \colon G \wr H \to H$ and $\tau = \tau_{G \wr H} \colon G \wr H \to G^{(H)}$ from section 4. For $g \in G \wr H$ we write $\mathsf{supp}(g)$ for $\mathsf{supp}(\tau(g))$.

A knapsack expression $E = h_0 g_1^{x_1} h_1 \cdots g_k^{x_k} h_k$ over $G \wr H$ is called *torsion-free* if for each $i \in [1,k]$, either $\sigma(g_i) = 1$ or $\sigma(g_i)$ has infinite order. A simple conjugation argument shows that it suffices to prove Theorem 5.3 and 5.4 for torsion-free knapsack expressions. For the rest of this section let us fix a torsion-free knapsack expression E over $G \wr H$. We can assume that $E = g_1^{x_1} g_2^{x_2} \cdots g_k^{x_k} g_{k+1}$ (note that if g has infinite order than also $c^{-1}gc$ has infinite order). We partition the set $V_E = \{x_1, \ldots, x_k\}$ of variables in E as $V_E = S \uplus M$, where $S = \{x_i \in V_E \mid \sigma(g_i) = 1\}$ and $M = \{x_i \in V_E \mid \mathrm{ord}(\sigma(g_i)) = \infty\}$. In this situation, the following notation will be useful. If $U = A \uplus B$ for a set of variables $U \subseteq V$ and $\mu \in \mathbb{N}^A$ and $\kappa \in \mathbb{N}^B$, then we write $\mu \oplus \kappa \in \mathbb{N}^U$ for the valuation with $(\mu \oplus \kappa)(x) = \mu(x)$ for $x \in A$ and $(\mu \oplus \kappa)(x) = \kappa(x)$ for $x \in B$.

Computing powers. A key observation in our proof is that in order to compute the group element $\tau(g^m)(h)$ (in the cursor intuition, this is the element labeling the point $h \in H$ in the wreath product element g^m) for $h \in H$ and $g \in G \wr H$, where $\sigma(g)$ has infinite order, one only has to perform at most $|\mathsf{supp}(g)|$ many multiplications in G, yielding a bound independent of m. We begin by introducing a partial order on H. Suppose $h \in H$ has infinite order (i.e. $\mathrm{ord}(h) = \infty$). For $h', h'' \in H$, we write $h' \preccurlyeq_h h''$ if there is an $n \geq 0$ with $h' = h^n h''$. Then, $\preccurlyeq_h$ is transitive. Moreover, since h has infinite order, $\preccurlyeq_h$ is also anti-symmetric and thus a partial order. Observe that if knapsack is decidable for H, given $h, h', h'' \in H$, we can decide whether h has infinite order and whether $h' \preccurlyeq_h h''$. This notion is used because for $g \in G \wr H$, the order $\preccurlyeq_{\sigma(g)}$ determines how to evaluate the mapping $\tau(g^m)$ at a certain element of H. We will sometimes want to multiply all elements a_i for $i \in I$ such that the order in which we multiply is specified by some linear order on I. If $(I, \leq)$ is a finite linearly ordered set with $I = \{i_1, \ldots, i_n\}$, $i_1 < i_2 < \ldots < i_n$, then we write $\prod_{i \in I}^{\leq} a_i$ for $\prod_{j=1}^{n} a_{i_j}$. If the order $\leq$ is clear from the context, we just write $\prod_{i \in I} a_i$.

Addresses. A central concept in our proof is that of an address. A solution to the equation $E = 1$ can be thought of as a sequence of instructions on how to walk through the Cayley graph of H and place elements of G at those nodes. Here, being a solution means that in the end, all the nodes contain the identity of G. In order to express that every node carries 1 in the end, we want to talk about at which points in the product $E = g_1^{x_1} g_2^{x_2} \cdots g_k^{x_k} g_{k+1}$ a particular node is visited. An address is a datum that contains just enough information about such a point to determine which element of G has been placed during that visit.

A pair (i, h) with $i \in [1, k+1]$, and $h \in H$ is called an *address* if $h \in \mathsf{supp}(g_i)$. The set of addresses of the expression E is denoted by A. Note that A is finite and computable. To each address (i, h), we associate the group element $\gamma(i, h) = g_i$ of the expression E.

A linear order on addresses. We will see that if a node is visited more than once, then (i) each time[1] it does so at a different address and (ii) the order of these visits only depends on the addresses. To capture the order of these visits, we define a linear order on addresses.

We partition $A = \bigcup_{i \in [1,k+1]} A_i$, where $A_i = \{(i, h) \mid h \in \mathsf{supp}(g_i)\}$ for $i \in [1, k+1]$. Then, for $a \in A_i$ and $a' \in A_j$ with $i < j$, we let $a < a'$. It remains to order addresses within each

[1] Here, we count two visits inside the same factor g_i, $i \in [1,k]$, with $\sigma(g_i) = 1$ as one visit.

A_i. Within A_{k+1}, we pick an arbitrary order. If $i \in [1,k]$ and $\sigma(g_i) = 1$, we also order A_i arbitrarily. Finally, if $i \in [1,k]$ and $\sigma(g_i)$ has infinite order, then we pick a linear order $\leq$ on A_i so that for $h, h' \in \mathsf{supp}(g_i)$, $h \preccurlyeq_{\sigma(g_i)} h'$ implies $(i,h) \leq (i,h')$. Note that this is possible since $\preccurlyeq_{\sigma(g_i)}$ is a partial order on H.

Cancelling profiles. In order to express that a solution for E yields the identity at every node of the Cayley graph of H, we need to compute the element of G that is placed after the various visits at a particular node. We therefore associate to each address an expression over G that yields the element placed during a visit at this address $a \in A$. In analogy to $\tau(g)$ for $g \in G \wr H$, we denote this expression by $\tau(a)$. If $a = (k+1, h)$, then we set $\tau(a) = \tau(g_{k+1})(h)$. Now, let $a = (i,h)$ for $i \in [1,k]$. If $\sigma(g_i) = 1$, then $\tau(a) = \tau(g_i)(h)^{x_i}$. Finally, if $\sigma(g_i)$ has infinite order, then $\tau(a) = \tau(g_i)(h)$.

This allows us to express the element of G that is placed at a node $h \in H$ if h has been visited with a particular set of addresses. To each subset $C \subseteq A$, we assign the expression $E_C = \prod_{a \in C} \tau(a)$, where the order of multiplication is given by the linear order on A. Observe that only variables in $S \subseteq \{x_1, \ldots, x_k\}$ occur in E_C. Therefore, given $\kappa \in \mathbb{N}^S$, we can evaluate $\kappa(E_C) \in G$. We say that $C \subseteq A$ is *κ-cancelling* if $\kappa(E_C) = 1$.

In order to record which sets of addresses can cancel simultaneously (meaning: for the same valuation), we use profiles. A *profile* is a subset of $\mathcal{P}(A)$ (the power set of A). A profile $P \subseteq \mathcal{P}(A)$ is said to be *κ-cancelling* if every $C \in P$ is κ-cancelling. A profile is *cancelling* if it is κ-cancelling for some $\kappa \in \mathbb{N}^S$.

Clusters. We also need to express that there is a node $h \in H$ that is visited with a particular set of addresses. To this end, we associate to each address $a \in A$ another expression $\sigma(a)$. As opposed to $\tau(a)$, the expression $\sigma(a)$ is over H and variables $M' = M \cup \{y_i \mid x_i \in M\}$. Let $a = (i,h) \in A$. When we define $\sigma(a)$, we will also include factors $\sigma(g_j)^{x_j}$ and $\sigma(g_j)^{y_j}$ where $\sigma(g_j) = 1$. However, since these factors do not affect the evaluation of the expression, this should be interpreted as leaving out such factors.

1. If $i = k+1$ then $\sigma(a) = \sigma(g_1)^{x_1} \cdots \sigma(g_k)^{x_k} h$.
2. If $i \in [1,k]$ then $\sigma(a) = \sigma(g_1)^{x_1} \cdots \sigma(g_{i-1})^{x_{i-1}} \sigma(g_i)^{y_i} h$.

We now want to express that when multiplying $g_1^{\nu(x_1)} \cdots g_k^{\nu(x_k)} g_{k+1}$, there is a node $h \in H$ such that the set of addresses with which one visits h is precisely $C \subseteq A$. In this case, we will call C a cluster. Let $\mu \in \mathbb{N}^M$ and $\mu' \in \mathbb{N}^{M'}$. We write $\mu' \sqsubset \mu$ if $\mu'(x_i) = \mu(x_i)$ for $x_i \in M$ and $\mu'(y_i) \in [0, \mu(x_i) - 1]$ for every $y_i \in M'$. We can now define the set of addresses at which one visits $h \in H$: For $h \in H$, let $A_{\mu,h} = \{a \in A \mid \mu'(\sigma(a)) = h \text{ for some } \mu' \in \mathbb{N}^{M'} \text{ with } \mu' \sqsubset \mu\}$. A subset $C \subseteq A$ is called a *μ-cluster* if $C \neq \emptyset$ and there is an $h \in H$ such that $C = A_{\mu,h}$. It can now be shown that if $\nu = \mu \oplus \kappa$ for $\kappa \in \mathbb{N}^S$ and $\mu \in \mathbb{N}^M$, evaluating $\tau(\nu(E))$ at a node $h \in H$ amounts to evaluating κ on the expression E_C where C is the μ-cluster $A_{\mu,h}$. In other words, we have $\tau(\nu(E))(h) = \kappa(E_C)$. From this, we obtain a characterization of solutions of the knapsack expression E.

▶ **Proposition 6.1.** *Let $\nu \in \mathbb{N}^{V_E}$ with $\nu = \mu \oplus \kappa$ for $\mu \in \mathbb{N}^M$ and $\kappa \in \mathbb{N}^S$. Then $\nu(E) = 1$ if and only if $\sigma(\nu(E)) = 1$ and there is a κ-cancelling profile P such that every μ-cluster is contained in P.*

This allows us to decompose a knapsack instance for $G \wr H$ into two tasks: determining which profiles are cancelling and finding a $\mu \in \mathbb{N}^M$ such that all μ-clusters are contained in a given profile. The first task can be reduced to solving exponent equation systems over G: For each profile $P \subseteq \mathcal{P}(A)$, let $K_P \subseteq \mathbb{N}^S$ be the set of all $\kappa \in \mathbb{N}^S$ such that P is κ-cancelling.

▶ **Lemma 6.2.** *Let P be a given profile. If $\mathrm{KP}(G^*)$ is decidable, then it is decidable whether $K_P \neq \emptyset$. If G is knapsack-semilinear, then $K_P \subseteq \mathbb{N}^S$ is effectively semilinear.*

For our second task, we employ the effective semilinearity of knapsack solution sets for H. Let $L_P \subseteq \mathbb{N}^M$ be the set of all $\mu \in \mathbb{N}^M$ such that every μ-cluster belongs to P.

▶ **Lemma 6.3.** *Let H be knapsack-semilinear. For every profile $P \subseteq \mathcal{P}(A)$, the set L_P is effectively semilinear.*

We can define L_P in Presburger arithmetic: In order to express that a given $C \subseteq A$ is a μ-cluster, we employ universal quantification to state that no other address $a \in A \setminus C$ is visited at the same node as the addresses in C. This leads to a Π_2-formula.

We can now prove Theorem 5.3 and 5.4. Let H be knapsack-semilinear and let $\mathrm{KP}(G^*)$ be decidable. Observe that for $\nu = \mu \oplus \kappa$, where $\mu \in \mathbb{N}^M$ and $\kappa \in \mathbb{N}^S$, the value of $\sigma(\nu(E))$ only depends on μ. The set $T \subseteq \mathbb{N}^M$ of all μ such that $\sigma(\nu(E)) = 1$ is effectively semilinear by knapsack-semilinearity of H. Proposition 6.1 tells us that $\mathsf{Sol}(E) = \bigcup_{P \subseteq \mathcal{P}(A)} K_P \oplus (L_P \cap T)$ and L_P is effectively semilinear by Lemma 6.3. This implies Theorem 5.3: We can decide solvability of E by checking, for each profile $P \subseteq \mathcal{P}(A)$, whether $K_P \neq \emptyset$ (which is decidable by Lemma 6.2) and whether $L_P \cap T \neq \emptyset$. Moreover, if G is knapsack-semilinear, then K_P and thus $\mathsf{Sol}(E)$ are effectively semilinear as well. This proves Theorem 5.4.

7 Complexity: Proof of Theorem 5.6

Throughout the section we fix a finitely generated group G. The goal of this section is to show that if G is abelian and non-trivial, then $\mathrm{KP}(G \wr \mathbb{Z})$ is NP-complete.

7.1 Periodic words over groups

With G^ω we denote the group of all mappings $f : \mathbb{N} \to G$ with the pointwise multiplication $(fg)(n) = f(n)g(n)$. The identity element is the mapping id with $\mathrm{id}(n) = 1$ for all $n \in \mathbb{N}$. If G is abelian, then also G^ω is abelian and we write $\sum_{i=1}^n f_i$ for $f_1 \cdots f_n$ with $f_i \in G^\omega$. A function $f \in G^\omega$ is *periodic with period* $q \geq 1$ if $f(k) = f(k+q)$ for all $k \geq 0$. Note that q is not assumed to be minimal. Let G^ρ be the set of all periodic functions from G^ω. With G^+ we denote the set of all tuples $(g_0, \ldots, g_{q-1})$ over G of arbitrary length $q \geq 1$. A periodic function $f \in G^\rho$ with period q can be specified by the tuple $(f(0), \ldots, f(q-1)) \in G^+$. Vice versa, a tuple $u = (g_0, \ldots, g_{q-1}) \in G^+$ defines the periodic function $f_u \in G^\omega$ with $f_u(n \cdot q + r) = g_r$ for $n \geq 0$ and $0 \leq r < q$. One can view this mapping as the sequence u^ω obtained by taking infinitely many repetitions of u. If f_1 is periodic with period q_1 and f_2 is periodic with period q_2, then $f_1 f_2$ is periodic with period $q_1 q_2$ (in fact, $\mathrm{lcm}(q_1, q_2)$). Hence, G^ρ forms a countable subgroup of G^ω. Note that G^ρ is not finitely generated: The subgroup generated by elements $f_i \in G^\rho$ with period q_i $(1 \leq i \leq n)$ contains only functions with period $\mathrm{lcm}(q_1, \ldots, q_n)$. For $n \geq 0$ let $G^\rho_n \leq G^\rho$ be the subgroup of all $f \in G^\rho$ with $f(k) = 1$ for all $0 \leq k \leq n-1$. The *uniform membership problem for subgroups* G^ρ_n, $\textsc{Membership}(G^\rho_*)$, is the following problem:

Input: Tuples $u_1, u_2, \ldots, u_n \in G^+$ and a binary encoded number m.
Question: Does the product $f_{u_1} f_{u_2} \cdots f_{u_n} \in G^\rho$ belong to G^ρ_m?

▶ **Theorem 7.1.** *For every finitely generated abelian group G, $\textsc{Membership}(G^\rho_*)$ can be solved in polynomial time.*

Proof. We use the additive notation for G^ω. Let $u_1, \dots, u_n \in G^+$, $q_i = |u_i|$, and $f = \sum_{i=1}^n f_{u_i}$. We show that if there exists a position m such that $f(m) \neq 0$, then there exists a position $m < \sum_{i=1}^n q_i$ such that $f(m) \neq 0$. This suffices to prove the theorem since the word problem for a finitely generated abelian group can be solved in polynomial time.

Let $m \geq \sum_{i=1}^n q_i$ and assume that $f(j) = 0$ for all j with $m - \sum_{i=1}^n q_i \leq j < m$. We show that $f(m) = 0$, which proves the above claim.

Note that $f_{u_i}(j) = f_{u_i}(j - q_i)$ for all $j \geq q_i$, $1 \leq i \leq n$. For $M \subseteq [1, n]$ let $q_M = \sum_{i \in M} q_i$. Moreover, for $1 \leq k \leq n$ let $\mathcal{M}_k = \{M \subseteq [1, n], |M| = k\}$. For $1 \leq k \leq n-1$ we get

$$\sum_{M \in \mathcal{M}_k} \sum_{i \in M} f_{u_i}(m - q_M) = \sum_{M \in \mathcal{M}_k} \sum_{i \in [1,n] \setminus M} -f_{u_i}(m - q_M) = \sum_{M \in \mathcal{M}_k} \sum_{i \in [1,n] \setminus M} -f_{u_i}(m - q_M - q_i)$$
$$= \sum_{i=1}^{n} \sum_{M \in \mathcal{M}_k, i \notin M} -f_{u_i}(m - q_{M \cup \{i\}}) = \sum_{M \in \mathcal{M}_{k+1}} \sum_{i \in M} -f_{u_i}(m - q_M).$$

We can write

$$f(m) = \sum_{i=1}^{n} f_{u_i}(m) = \sum_{i=1}^{n} f_{u_i}(m - q_i) = \sum_{M \in \mathcal{M}_1} \sum_{i \in M} f_{u_i}(m - q_M).$$

From the above identities we get by induction:

$$f(m) = \pm \sum_{M \in \mathcal{M}_n} \sum_{i \in M} f_{u_i}(m - q_M) = \pm \sum_{i \in [1,n]} f_{u_i}(m - q_{[1,n]}) = \pm f(m - \sum_{i=1}^{n} q_i) = 0.$$

This proves the claim and hence the theorem. ◀

7.2 Automata for Cayley representations

The main technical result of this section is:

▶ **Proposition 7.2.** *Let G be a finitely generated abelian group. If* $\textsc{ExpEq}(G) \in \mathsf{NP}$ *and* $\textsc{Membership}(G^\rho_*) \in \mathsf{NP}$*, then also* $\mathrm{KP}(G \wr \mathbb{Z}) \in \mathsf{NP}$.

We start with some definitions. An interval $[a, b] \subseteq \mathbb{Z}$ *supports* an element $(f, d) \in G \wr \mathbb{Z}$ if $\{0, d\} \cup \mathsf{supp}(f) \subseteq [a, b]$. If $(f, d) \in G \wr \mathbb{Z}$ is a product of length n over the generators, then the minimal interval $[a, b]$ which supports (f, d) satisfies $b - a \leq n$. A knapsack expression $E = v_0 u_1^{x_1} v_1 \cdots u_k^{x_k} v_k$ is called *rigid* if each u_i evaluates to an element $(f_i, 0) \in G \wr \mathbb{Z}$. Intuitively, the movement of the cursor is independent from the values of the variables x_i up to repetition of loops. In particular, every variable-free expression is rigid.

In the following we define the so called Cayley representation of a rigid knapsack expression. This is a finite word, where every symbol is a marked knapsack expression over G. A marked knapsack expression over G is of the form E, $\overline{E}$, $\underline{E}$, or $\underline{\overline{E}}$, where E is a knapsack expression over G. We say that $\overline{E}$ and $\underline{\overline{E}}$ (resp., $\underline{E}$ and $\underline{\overline{E}}$) are top-marked (resp., bottom-marked).

Let $E = v_0 u_1^{x_1} v_1 \cdots u_k^{x_k} v_k$ be a rigid knapsack expression over $G \wr \mathbb{Z}$. For an assignment ν let $(f_\nu, d) \in G \wr \mathbb{Z}$ be the element to which $\nu(E)$ evaluates, i.e. $(f_\nu, d) = \nu(E)$. Note that d does not depend on ν. Because of the rigidity of E, there is an interval $[a, b] \subseteq \mathbb{Z}$ that supports (f_ν, d) for all assignments ν. For each $j \in [a, b]$ let E_j be a knapsack expression over G with the variables $x_1, \dots, x_k$ such that $f_\nu(j) = \nu(E_j)$ for all assignments ν. Then we call the formal expression

$$r = \begin{cases} E_a\, E_{a+1} \cdots E_{-1}\, \overline{E_0}\, E_1 \cdots E_{d-1}\, \underline{E_d}\, E_{d+1} \cdots E_b & \text{if } d > 0 \\ E_a\, E_{a+1} \cdots E_{-1}\, \underline{\overline{E_0}}\, E_1 \cdots E_b & \text{if } d = 0 \\ E_a\, E_{a+1} \cdots E_{d-1}\, \underline{E_d}\, E_{d+1} \cdots E_{-1}\, \overline{E_0}\, E_1 \cdots E_b & \text{if } d < 0 \end{cases}.$$

-1	0	1	2	3	4	5	6	7	8	9	10	11	12
a^x	$\overline{a^x}$	b^x											
	$\overline{1}$	$\underline{1}$											
		$\overline{b^y}$	b^y	b^y									
	b	$\overline{a}$	b^{-1}	$\underline{a}$	a								
			b	$\overline{a}$	b^{-1}	$\underline{a}$	a						
					b	$\overline{a}$	b^{-1}	$\underline{a}$	a				
							b	$\overline{a}$	b^{-1}	$\underline{a}$	a		
									b	$\overline{a}$	b^{-1}	$\underline{a}$	a
a^x	$\overline{a^x b}$	$b^x b^y a$	b^y	$b^y a^2$	a	a^2	a	a^2	a	a^2	ab^{-1}	$\underline{a}$	a

Figure 1 Cayley representation.

a *Cayley representation* of E (or E is *represented* by r). Formally, a Cayley representation r is a sequence of marked knapsack expressions, and the length of this sequence is denote by $|r|$. In our examples, we separate for better readability consecutive marked knapsack expressions in r by commas. By the above definition, r depends on the chosen supporting interval $[a, b]$. However, compared to the representation of the minimal supporting interval, any other Cayley representation differs only by adding 1's (i.e., trivial knapsack expressions over G) at the left and right end of r.

A Cayley representation of E records for each point in $\mathbb{Z}$ an expression that describes which element will be placed at that point. Multiplying an element of $G \wr \mathbb{Z}$ always begins at a particular cursor position; in a Cayley representation, the marker on top specifies the expression that is placed at the cursor position in the beginning. Moreover, a Cayley representation describes how the cursor changes when multiplying $\nu(E)$: The marker on the bottom specifies where the cursor is located in the end.

▶ **Example 7.3.** Consider the wreath product $F_2 \wr \mathbb{Z}$, where F_2 is the free group generated by $\{a, b\}$ and $\mathbb{Z}$ is generated by t, and the rigid knapsack expression $E = u_1^x u_2 u_3^y u_4^5$ with $u_1 = at^{-1}at^2bt^{-1}$ (represented by $a\,\overline{a}\,b$), $u_2 = t$ (represented by $\overline{1}\,\underline{1}$), $u_3 = btbtbt^{-2}$ (represented by $\overline{b}\,b\,b$), and $u_4 = at^{-1}bt^2b^{-1}tatat^{-1}$ (represented by $b\,\overline{a}\,b^{-1}\,\underline{a}\,a$).

A Cayley representation of u_1^x is $a^x\,\overline{\underline{a^x}}\,b^{-1}$ and a Cayley representation of u_3^y is $\overline{\underline{b^y}}\,b^y\,b^y$. The diagram in Figure 1 illustrates how to compute a Cayley representation r of E, which is shown in the bottom line. Here, we have chosen the supporting interval minimal. Note that if we replace the exponents 5 in u_4^5 by a larger number, then we only increase the number of repetitions of the factor a, a^2 in the Cayley representation.

Let E be an arbitrary knapsack expression over $G \wr \mathbb{Z}$. We can assume that E has the form $u_1^{x_1} \cdots u_k^{x_k} u_{k+1}$. Let X_0 be the set of all variables x_i where u_i evaluates to an element $(f, 0) \in G \wr \mathbb{Z}$, and let $X_1 = \{x_1, \ldots, x_k\} \setminus X_0$. For a partial assignment $\nu\colon X_1 \to \mathbb{N}$ we obtain a rigid knapsack expression E_ν by replacing in E every variable $x_i \in X_1$ by $\nu(x_i)$. A set R of Cayley representations is a *set representation* of E if

- for each assignment $\nu\colon X_1 \to \mathbb{N}$ there exists $r \in R$ such that r represents E_ν,
- for each $r \in R$ there exists an assignment $\nu\colon X_1 \to \mathbb{N}$ such that r represents E_ν and $\nu(x) \leq |r|$ for all $x \in X_1$.

▶ **Example 7.4.** Consider the non-rigid knapsack expression $E' = u_1^x u_2 u_3^y u_4^z$ over $F_2 \wr \mathbb{Z}$, where u_1, u_2, u_3, u_4 are taken from Example 7.3. We have $X_0 = \{x, y\}$ and $X_1 = \{z\}$. A set representation R of E' consists of the following Cayley representations: $a^x, \overline{a^x}, \underline{b^x b^y}, b^y, b^y$

for $\nu(z) = 0$, $a^x, \overline{a^x b}, b^x b^y a, b^y b^{-1}, \underline{b^y a}, a$ for $\nu(z) = 1$, and

$$a^x, \overline{a^x b}, b^x b^y a, b^y, b^y a^2, (a, a^2)^{\nu(z)-2}, ab^{-1}, \underline{a}, a \quad \text{for } \nu(z) \geq 2.$$

Only finitely many different marked knapsack expressions appear in this set representation R, and R is clearly a regular language over the finite alphabet consisting of this finitely many marked knapsack expressions.

We can now sketch the proof of Proposition 7.2. The main idea is to construct a nondeterministic finite automaton (NFA) that accepts a set representation of $E = u_1^{x_1} \cdots u_k^{x_k} u_{k+1}$. Let $n = |E|$. First, we compute polynomial-size NFAs $\mathcal{A}_i$ ($i \in [1, k+1]$), where $\mathcal{A}_i$ accepts a set representation of $u_i^{x_i}$ (or u_{k+1} for $i = k+1$). For u_{k+1} and expressions $u_i^{x_i}$ with $x_i \in X_0$ these set representations are singletons and the construction of $\mathcal{A}_i$ is straightforward, see e.g. Example 7.3. For expressions $u_i^{x_i}$ with $x_i \in X_1$ one has to construct an NFA that accepts a set containing a Cayley representation of every u_i^m (a variable-free knapsack expression over G) for $m \geq 0$. The main observation is that all these Cayley representations are periodic (except for a short prefix and suffix) with the same period.

From the NFAs $\mathcal{A}_i$ one obtains an NFA $\mathcal{A}$ accepting a set representation of E using a simple product construction. This NFA $\mathcal{A}$ has exponential size in n, so we cannot construct it. However, its exponential size bound on $\mathcal{A}$ yields that $E = 1$ has a solution if and only if there exists a solution ν such that $\nu(x)$ is exponentially bounded in n for all $x \in X_1$. Since each $\mathcal{A}_i$ accepts a set representation of $u_i^{x_i}$, $i \in [1, k]$ or of u_{k+1}, this implies that solvability of E is witnessed by words $\alpha_i \in L(\mathcal{A}_i)$ for $i \in [1, k+1]$ whose length is bounded exponentially in n. The periodic nature of the words α_i allows to represent these words in polynomial space as a concatenation of powers β^m for binary encoded numbers m. We guess such representations of the α_i.

It remains to verify that the guessed words α_i indeed witness a solution of $E = 1$. This means that there exists a valuation $\nu\colon X_0 \to \mathbb{N}$ such that for every position p the $(k+1)$-tuple consisting of the p-th entries of the α_i evaluates to 1 under ν. There exist only polynomially many positions p where an expression u^x with $x \in X_0$ occurs in some α_i. Thus, we can construct from all these positions an instance of $\textsc{ExpEq}(G)$. The remaining pieces of the α_i only contain group elements from G and are periodic. The question, whether they cancel at all remaining positions is an instance of $\textsc{Membership}(G_*^\rho)$.

Proposition 7.2 yields the NP upper bound for Theorem 5.6: If G is a finitely generated abelian group, then $\textsc{ExpEq}(G)$ corresponds to the solvability problem for linear equation systems over the integers, possibly with modulo-constraints. This problem is well known to be in NP. Moreover, $\textsc{Membership}(G_*^\rho)$ can be solved in polynomial time by Theorem 7.1.

It remains to prove the NP-hardness part of Theorem 5.6. Using a reduction from 3-dimensional matching, one can show the following [4]:

▶ **Theorem 7.5.** *If G is non-trivial and H contains an element of infinite order, then knapsack and subset sum for $G \wr H$ are NP-hard.*

8 Open problems

The main open problem is to characterize those G and H for which $\mathrm{KP}(G \wr H)$ is decidable. Concerning complexity, we are confident that our NP upper bound for $\mathrm{KP}(G \wr \mathbb{Z})$, where G is finitely generated abelian, can be extended to $\mathrm{KP}(G \wr H)$, where H is a finitely generated free group or $\mathbb{Z}^n$ for some $n \geq 0$. Another question is whether the assumption on G being abelian can be weakened. In particular, we want to investigate whether polynomial time algorithms exist for $\textsc{Membership}(G_*^\rho)$ for certain non-abelian groups G.

References

1 T. C. Davis and A. Yu. Olshanskii. Subgroup distortion in wreath products of cyclic groups. *Journal of Pure and Applied Algebra*, 215(12):2987–3004, 2011.

2 M. Elberfeld, A. Jakoby, and T. Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:128, 2011.

3 E. Frenkel, A. Nikolaev, and A. Ushakov. Knapsack problems in products of groups. *Journal of Symbolic Computation*, 74:96–108, 2016.

4 M. Ganardi, D. König, M. Lohrey, and G. Zetzsche. Knapsack problems for wreath products. *CoRR*, abs/1709.09598, 2017. URL: `http://arxiv.org/abs/1709.09598`.

5 S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966. `doi:10.2140/pjm.1966.16.285`.

6 C. Haase. *On the complexity of model checking counter automata*. PhD thesis, University of Oxford, St Catherine's College, 2011.

7 R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

8 D. König, M. Lohrey, and G. Zetzsche. Knapsack and subset sum problems in nilpotent, polycyclic, and co-context-free groups. In *Algebra and Computer Science*, volume 677 of *Contemporary Mathematics*, pages 138–153. American Mathematical Society, 2016.

9 J. Lehnert and P. Schweitzer. The co-word problem for the Higman-Thompson group is context-free. *Bulletin of the London Mathematical Society*, 39(2):235–241, 2007.

10 M. Lohrey, B. Steinberg, and G. Zetzsche. Rational subsets and submonoids of wreath products. *Information and Computation*, 243:191–204, 2015.

11 M. Lohrey and G. Zetzsche. Knapsack in graph groups, HNN-extensions and amalgamated products. In Nicolas Ollinger and Heribert Vollmer, editors, *Proc. of the 33rd International Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, volume 47 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

12 M. Lohrey and G. Zetzsche. The complexity of knapsack in graph groups. In *Proceedings of the 34th Symposium on Theoretical Aspects of Computer Science, STACS 2017*, volume 66 of *LIPIcs*, pages 52:1–52:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

13 A. Miasnikov, S. Vassileva, and A. Weiß. The conjugacy problem in free solvable groups and wreath products of abelian groups is in TC^0. In *Computer Science – Theory and Applications – 12th International Computer Science Symposium in Russia, CSR 2017, Proceedings*, volume 10304 of *Lecture Notes in Computer Science*, pages 217–231. Springer, 2017.

14 A. Mishchenko and A. Treier. Knapsack problem for nilpotent groups. *Groups Complexity Cryptology*, 9(1):87–98, 2017.

15 A. Myasnikov, A. Nikolaev, and A. Ushakov. Knapsack problems in groups. *Mathematics of Computation*, 84:987–1016, 2015.

16 A. Nikolaev and A. Ushakov. Subset sum problem in polycyclic groups. *Journal of Symbolic Computation*, 84:84–94, 2018.

17 C. Sims. *Computation with finitely presented groups*. Cambridge University Press, 1994.

18 M. Wilhelm. On a theorem of Marshall Hall. *Annals of Mathematics. Second Series*, 40:764–768, 1939.

On Structural Parameterizations of the Bounded-Degree Vertex Deletion Problem

Robert Ganian[1]
Algorithms and Complexity Group, TU Wien, Vienna, Austria

Fabian Klute
Algorithms and Complexity Group, TU Wien, Vienna, Austria

Sebastian Ordyniak
Algorithms and Complexity Group, TU Wien, Vienna, Austria

Abstract

We study the parameterized complexity of the Bounded-Degree Vertex Deletion problem (BDD), where the aim is to find a maximum induced subgraph whose maximum degree is below a given degree bound. Our focus lies on parameters that measure the structural properties of the input instance. We first show that the problem is W[1]-hard parameterized by a wide range of fairly restrictive structural parameters such as the feedback vertex set number, pathwidth, treedepth, and even the size of a minimum vertex deletion set into graphs of pathwidth and treedepth at most three. We thereby resolve the main open question stated in Betzler, Bredereck, Niedermeier and Uhlmann (2012) concerning the complexity of BDD parameterized by the feedback vertex set number. On the positive side, we obtain fixed-parameter algorithms for the problem with respect to the decompositional parameter treecut width and a novel problem-specific parameter called the core fracture number.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis, Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Bounded-degree Vertex Deletion, Feedback Vertex Set, Parameterized Algorithms, Treecut Width

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.33

Funding This work was supported by the Austrian Science Fund (FWF), project P26696.

1 Introduction

This paper studies the Bounded-Degree Vertex Deletion problem (BDD): given an undirected graph G, a degree bound d, and a limit ℓ, determine whether it is possible to delete at most ℓ vertices from G in order to obtain a graph of maximum degree at most d. Aside from being a natural generalization of the classical Vertex Cover problem, BDD has found applications in areas such as computational biology [17] and is the dual problem of the so-called *s-Plex Detection* problem in social network analysis [30, 3, 31, 35].

It is not surprising that the complexity of BDD and several of its variants has been studied extensively by the theory community in the past years [5, 4, 7, 6, 11, 26, 34, 35]. Since the problem is NP-complete in general, it is natural to ask under which conditions does the problem become tractable. In this direction, the *parameterized complexity* paradigm [13, 33, 9] allows a more refined analysis of the problem's complexity than classical complexity. In the

[1] Robert Ganian is also affiliated with FI MU, Brno, Czech Republic.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 33; pp. 33:1–33:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

parameterized setting, we associate each instance with a numerical parameter k and are most often interested in the existence of a *fixed-parameter algorithm*, i.e., an algorithm solving the problem in time $f(k) \cdot |V(G)|^{\mathcal{O}(1)}$ for some computable function f. Parameterized problems admitting such an algorithm belong to the class FPT; on the other hand, parameterized problems that are hard for the complexity class W[1] or W[2] do not admit fixed-parameter algorithms (under standard complexity assumptions).

In general, there exist two notable approaches for selecting parameters: a parameter may either originate from the formulation of the problem itself (often called *natural parameters*), or rather from the structure of the input graph (so-called *structural parameters*, most prominently represented by the decomposition-based parameter *treewidth* **tw**). The parameterized complexity of BDD has already been studied extensively through the lens of natural parameters (especially d and ℓ). In particular, BDD is known to be FPT when parameterized by $d + \ell$ [34, 17, 31], W[2]-hard when parameterized only by ℓ [17], and NP-complete when parameterized only by d (as witnessed by the case of $d = 0$, i.e., Vertex Cover). The complexity of BDD is also fairly well understood when considering combinations of natural and structural parameters: it is FPT when parameterized by $\mathbf{tw} + d$ due to Courcelle's Theorem [8] and has been shown to be FPT when parameterized by $\mathbf{tw} + \ell$ [5].

Given the above, it is fairly surprising that the problem has remained fairly unexplored when viewed through the lens of structural parameters only, i.e., in the case where we impose no restrictions on the problem formulation itself but only on the structure of the graph. BDD was shown not to be FPT when parameterized by treewidth [5], complementing the previous $\mathcal{O}(n^{\mathbf{tw}+1})$ algorithm of Dessmark et al. [11]. The only structural parameter which is known to make the problem fixed-parameter tractable is the *feedback edge set number*, i.e., the minimum number of edges whose deletion results in a forest [5].

Contribution

The goal of this paper is to provide new insight into the complexity of BDD parameterized by the structure of the input graph. Our first main result shows that BDD is W[1]-hard parameterized by the *feedback vertex set number*, i.e., the minimum number of vertices whose deletion results in a forest. This resolves the main open question in [5]. Interestingly, our result is significantly stronger since we show that hardness even applies in the case that the remaining parts, after deleting the feedback vertex set, are trees of height three. This rules out fixed-parameter algorithms w.r.t. most of the remaining "classical" decomposition-based structural parameters such as *pathwidth* and *treedepth* [32] as well as w.r.t. the *vertex deletion distance* [19, 32] to bounded pathwidth, treedepth, and treewidth. On the way to our hardness result we show hardness for several multidimensional variants of the classical subset sum problem parameterized by the number of dimensions, which we believe are interesting on their own.

In light of the above, it is natural to ask whether there exist natural decomposition-based parameters for which BDD is fixed-parameter tractable. Our main algorithmic result answers this question affirmatively: we obtain a fixed-parameter algorithm utilizing the recently introduced structural parameter called *treecut width*. The importance of treecut width is that it plays a similar role with respect to the fundamental graph operation of immersion as the graph parameter *treewidth* plays with respect to the minor operation [36, 29]. Up to now, only a handful of problems are known to be FPT when parameterized by treecut width but W[1]-hard when parameterized by treewidth [20]. Furthermore, unlike previously known algorithms using treecut width, this is the first of its kind which does not use an Integer Linear Programming formulation but instead relies purely on combinatorial arguments.

Our second algorithmic result focuses on structural parameters which are not based on any particular decomposition of the graph, but instead measure the "vertex-deletion distance" to a certain graph property. Such structural parameters have been successfully used in the past for a plethora of other difficult problems [19, 22, 27, 15, 14, 21]. In this context and taking into account the strong lower bounds obtained in Section 3, we introduce a structural parameter which is specifically tailored to BDD and which we call the *core fracture number*. Roughly speaking, the core fracture number k is the vertex deletion distance to a graph where each connected component only contains at most k vertices which exceed the degree bound d. We show that computing the core fracture number is FPT which in turn gives rise to a fixed-parameter algorithm for BDD; the latter is achieved by identifying and formalizing a *type-aggregation condition*, allowing for an encoding of the problem into an Integer Linear Program with a controlled number of integer variables. This also resolves the question from [5] if BDD is FPT parametrized by vertex cover.

Finally, we exclude the existence of a *polynomial kernel* [13, 9] for BDD parameterized by the treecut width and core fracture number, and compare the two parameters in Section 5.

2 Preliminaries

2.1 Basic Notation

We use standard terminology for graph theory, see for instance [12]. All graphs except for those used to compute the torso-size in Subsection 2.3 are simple; the multigraphs used in Subsection 2.3 have loops, and each loop increases the degree of the vertex by 2.

Let G be a graph. We denote by $V(G)$ and $E(G)$ its vertex and edge set, respectively. For a vertex $v \in V(G)$, let $N_G(v) = \{y \in V(G) : vy \in E(G)\}$, $N_G[v] = N_G(v) \cup \{v\}$, and $\deg_G(v)$ denote its open neighborhood, closed neighborhood, and degree, respectively. For a subset $X \subseteq V(G)$, the (open) neighborhood $N_G(X)$ of X is defined as $\bigcup_{x \in X} N(x) \setminus X$. The set $N_G[X]$ refers to the closed neighborhood of X defined as $N_G(X) \cup X$. We refer to the set $N_G(V(G) \setminus X)$ as $\partial_G(X)$; this is the set of vertices in X which have a neighbor in $V(G) \setminus X$. We omit the lower index G, if G is clear from the context. For a vertex set A, we use $G - A$ to denote the graph obtained from G by deleting all vertices in A. We use $[i]$ to denote the set $\{0, 1, \ldots, i\}$. For completeness, we provide a formal definition of our problem of interest below.

Bounded-Degree Vertex Deletion (BDD)	
Input:	An undirected graph $G = (V, E)$ and integers $d \geq 0$ and $\ell \geq 0$.
Question:	Is there a subset $V' \subseteq V$ with $\lvert V' \rvert \leq \ell$ whose removal from G yields a graph in which each vertex has degree at most d?

2.2 Parameterized Complexity

A *parameterized problem* $\mathcal{P}$ is a subset of $\Sigma^* \times \mathbb{N}$ for some finite alphabet Σ. Let $L \subseteq \Sigma^*$ be a classical decision problem for a finite alphabet, and let p be a non-negative integer-valued function defined on Σ^*. Then L *parameterized by* p denotes the parameterized problem $\{ (x, p(x)) \mid x \in L \}$ where $x \in \Sigma^*$. For a problem instance $(x, k) \in \Sigma^* \times \mathbb{N}$ we call x the main part and k the parameter. A parameterized problem $\mathcal{P}$ is *fixed-parameter tractable* (FPT in short) if a given instance (x, k) can be solved in time $\mathcal{O}(f(k) \cdot p(|x|))$ where f is an arbitrary computable function of k and p is a polynomial function; we call algorithms running in this time *fixed-parameter algorithms*. We refer the reader to [13] for more details on parameterized complexity.

Parameterized complexity classes are defined with respect to *fpt-reducibility*. A parameterized problem P is *fpt-reducible* to Q if in time $f(k) \cdot |x|^{O(1)}$, one can transform an instance (x, k) of $\mathcal{P}$ into an instance (x', k') of $\mathcal{Q}$ such that $(x, k) \in \mathcal{P}$ if and only if $(x', k') \in \mathcal{Q}$, and $k' \leq g(k)$, where f and g are computable functions depending only on k. Central to parameterized complexity is the following hierarchy of complexity classes, defined by the closure of canonical problems under fpt-reductions: $\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \cdots \subseteq \mathsf{XP}$. All inclusions are believed to be strict. In particular, $\mathsf{FPT} \neq \mathsf{W}[1]$ under the Exponential Time Hypothesis [23].

The class $\mathsf{W}[1]$ is the analog of NP in parameterized complexity. A major goal in parameterized complexity is to distinguish between parameterized problems which are in FPT and those which are $\mathsf{W}[1]$-hard, i.e., those to which every problem in $\mathsf{W}[1]$ is fpt-reducible. There are many problems shown to be complete for $\mathsf{W}[1]$, or equivalently $\mathsf{W}[1]$-complete, including the Multi-Colored Clique (MCC) problem [13].

2.3 Treecut Width

The notion of treecut decompositions was first proposed by Wollan [36], see also [29]. A family of subsets $X_1, \ldots, X_k$ of X is a *near-partition* of X if they are pairwise disjoint and $\bigcup_{i=1}^{k} X_i = X$, allowing the possibility of $X_i = \emptyset$.

▶ **Definition 1.** A *treecut decomposition* of G is a pair $(T, \mathcal{X})$ which consists of a rooted tree T and a near-partition $\mathcal{X} = \{X_t \subseteq V(G) : t \in V(T)\}$ of $V(G)$. A set in the family $\mathcal{X}$ is called a *bag* of the treecut decomposition.

For any node t of T other than the root r, let $e(t) = ut$ be the unique edge incident to t on the path to r. Let T_u and T_t be the two connected components in $T - e(t)$ which contain u and t, respectively. Note that $(\bigcup_{q \in T_u} X_q, \bigcup_{q \in T_t} X_q)$ is a near-partition of $V(G)$, and we use $\mathbf{cut}(t)$ to denote the set of edges with one endpoint in each part. We define the *adhesion* of t ($\mathbf{adh}_T(t)$ or $\mathbf{adh}(t)$ in brief) as $|\mathbf{cut}(t)|$; if t is the root, we set $\mathbf{adh}_T(t) = 0$ and $\mathbf{cut}(t) = \emptyset$.

The *torso* of a treecut decomposition $(T, \mathcal{X})$ at a node t, written as H_t, is the graph obtained from G as follows. If T consists of a single node t, then the torso of $(T, \mathcal{X})$ at t is G. Otherwise let $T_1, \ldots, T_\ell$ be the connected components of $T - t$. For each $i = 1, \ldots, \ell$, the vertex set $Z_i \subseteq V(G)$ is defined as the set $\bigcup_{b \in V(T_i)} X_b$. The torso H_t at t is obtained from G by *consolidating* each vertex set Z_i into a single vertex z_i (this is also called *shrinking* in the literature). Here, the operation of consolidating a vertex set Z into z is to substitute Z by z in G, and for each edge e between Z and $v \in V(G) \setminus Z$, adding an edge zv in the new graph. We note that this may create parallel edges.

The operation of *suppressing* (also called *dissolving* in the literature) a vertex v of degree at most 2 consists of deleting v, and when the degree is two, adding an edge between the neighbors of v. Given a connected graph G and $X \subseteq V(G)$, let the *3-center* of (G, X) be the unique graph obtained from G by exhaustively suppressing vertices in $V(G) \setminus X$ of degree at most two. Finally, for a node t of T, we denote by $\tilde{H}_t$ the 3-center of (H_t, X_t), where H_t is the torso of $(T, \mathcal{X})$ at t. Let the *torso-size* $\mathbf{tor}(t)$ denote $|\tilde{H}_t|$.

▶ **Definition 2.** The width of a treecut decomposition $(T, \mathcal{X})$ of G is $\max_{t \in V(T)} \{\mathbf{adh}(t), \mathbf{tor}(t)\}$. The treecut width of G, or $\mathbf{tcw}(G)$ in short, is the minimum width of $(T, \mathcal{X})$ over all treecut decompositions $(T, \mathcal{X})$ of G.

We conclude this subsection with some notation related to treecut decompositions. Given a tree node t, let T_t be the subtree of T rooted at t. Let $Y_t = \bigcup_{b \in V(T_t)} X_b$, and let G_t denote

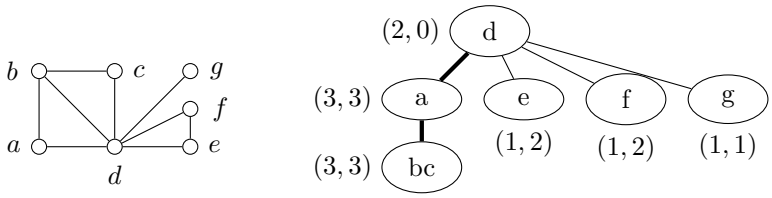


Figure 1 A graph G and a width-3 treecut decomposition of G, including the torso-size (left value) and adhesion (right value) of each node.

the induced subgraph $G[Y_t]$. The *depth* of a node t in T is the distance of t from the root r. The vertices of $\partial_t = \partial_G(Y_t)$ are called the *border* at node t. A node $t \neq r$ in a rooted treecut decomposition is *thin* if $\mathbf{adh}(t) \leq 2$ and *bold* otherwise. For a node t, we let B_t and A_t denote the set of children of t which are thin and bold, respectively.

While it is not known how to compute optimal treecut decompositions efficiently, there exists a fixed-parameter 2-approximation algorithm which fully suffices for our purposes.

▶ **Theorem 3** ([24]). *There exists an algorithm that takes as input an n-vertex graph G and integer k, runs in time $2^{\mathcal{O}(k^2 \log k)} n^2$, and either outputs a treecut decomposition of G of width at most $2k$ or correctly reports that $\mathbf{tcw}(G) > k$.*

A treecut decomposition $(T, \mathcal{X})$ is *nice* if it satisfies the following condition for every thin node $t \in V(T)$: $N(Y_t) \cap \bigcup_{b \text{ is a sibling of } t} Y_b = \emptyset$. The intuition behind nice treecut decompositions is that we restrict the neighborhood of thin nodes in a way which facilitates dynamic programming.

▶ **Lemma 4** ([20]). *There exists a cubic-time algorithm which transforms any rooted treecut decomposition $(T, \mathcal{X})$ of G into a nice treecut decomposition of the same graph, without increasing its width or number of nodes.*

The following property of nice treecut decompositions will be crucial for our algorithm.

▶ **Lemma 5** ([20]). *Let t be a node in a nice treecut decomposition of width k. Then $|A_t| \leq 2k + 1$.*

We refer to previous work [20] for a comparison of treecut width to other parameters.

3 Hardness Results

In this section we show that BDD is W[1]-hard parameterized by a vertex deletion set to trees of height at most three, i.e., a subset D of the vertices of the graph such that every component in the graph, after removing D, is a tree of height at most three. On the way towards this result, we provide hardness results for several interesting versions of the multidimensional subset sum problem (parameterized by the number of dimensions) which we believe are interesting in their own right. In particular, we note that the hardness results also hold for the well-known and more general multidimensional knapsack problem [18].

Our first auxiliary result shows hardness for the following problem.

Multidimensional Subset Sum (MSS)	
Input:	An integer k, a set $S = \{s_1, \ldots, s_n\}$ of item-vectors with $s_i \in \mathbb{N}^k$ for every i with $1 \leq i \leq n$ and a target vector $t \in \mathbb{N}^k$.
Parameter:	k
Question:	Is there a subset $S' \subseteq S$ such that $\sum_{s \in S'} s = t$?

▶ **Lemma 6.** MSS *is* W[1]*-hard even if all integers in the input are given in unary.*

Proof sketch. The proof is by a parameterized reduction from the well-known W[1]-hard Multicolored Clique (MCC) problem [13]: given a k-partite graph G with partition $V_1, \ldots, V_k$, decide whether G contains a clique of size k. For an instance $\mathcal{I} = (G, k)$ of MCC we construct an equivalent instance $\mathcal{I}' = (2\binom{k}{2} + k, S, t)$ of MSS in polynomial time, as follows. For every $v \in V(G)$ we construct one item-vector s_v in S and for every $e \in E(G)$ one item-vector s_e. Furthermore, we impose the following requirements on every solution $S' \subseteq S$ of $\mathcal{I}'$: (1) exactly one vector s_v with $v \in V_i$ is contained in S' for every i with $1 \leq i \leq k$, (2) exactly one vector s_e, with e being an edge between V_i and V_j, is contained in S' for every i and j with $1 \leq i < j \leq k$, and (3) for every edge e with $s_e \in S'$ and endpoints $v_i \in V_i$, $v_j \in V_j$ we find $s_{v_i}, s_{v_j} \in S'$. To ensure (1), the target vector has k entries with value one and every vector s_v with $v \in V_i$ has value one at the i-th of those entries. Property (2) is ensured in a similar way by using $\binom{k}{2}$ entries with value one in the target vector. To ensure Property (3), we assign to every vertex v of G a unique number $\mathcal{S}(v)$ from a Sidon sequence $\mathcal{S}$ of length $|V(G)|$ [16]. A Sidon sequence is a sequence of natural numbers such that the sum of each pair of numbers is unique; it can be shown that it is possible to construct such sequences whose maximum value is bounded by a polynomial in its length [1, 16]. The target vector then contains one additional entry $I(i, j)$ for every i and j with $1 \leq i < j \leq k$ with value $\max_2(\mathcal{S}) + 1$, where $\max_2(\mathcal{S})$ is the maximum sum of any two numbers in $\mathcal{S}$. Moreover, every vector s_v for $v \in V(G)$ has value $\mathcal{S}(v)$ at every entry $I(l, r)$ with $l = i$ or $r = i$ and similarly every vector s_e for an edge e between V_i and V_j has value $(\max_2(\mathcal{S}) + 1) - (\mathcal{S}(u) + \mathcal{S}(v))$ at entry $I(i, j)$. Then, because $\mathcal{S}$ is a Sidon sequence, it holds that the $I(i, j)$-th entry of $\sum_{s \in S'} s$ for a solution S' is equal to the $I(i, j)$-th entry of t if and only if the endpoints of the unique edge chosen between V_i and V_j are equal to the unique vertices v_i and v_j chosen in V_i and V_j, respectively. ◀

The proof of the above lemma also implies hardness for the following slightly adapted version of MSS, which we call the Restricted Multidimensional Subset Sum (RMSS) problem. For RMSS an additional integer k' is given (which will be part of the parameter) and we ask for a solution of the MSS problem of size exactly k'. Before presenting our hardness result for BDD, we need to show hardness for the following more relaxed version of RMSS, which we call the Multidimensional Relaxed Subset Sum (MRSS) problem. For MRSS both the input as well as the parameters are the same as in the case of RMSS however one now asks whether there is a subset $S' \subseteq S$ with $|S'| \leq k'$ such that $\sum_{s \in S'} s \geq t$.

▶ **Lemma 7.** MRSS *is* W[1]*-hard even if all integers in the input are given in unary.*

We are now ready to show our main hardness result for BDD using a reduction from MRSS.

▶ **Theorem 8.** BDD *is* W[1]*-hard parameterized by the size of a vertex deletion set into trees of height at most* 3*.*

Proof Sketch. We prove the theorem by a parameterized reduction from MRSS. Namely, given an instance $\mathcal{I} = (k, S, t, k')$ of MRSS we construct an equivalent instance $\mathcal{I}' = (G, d, \ell)$ of BDD such that G has a FVS D of size $k \cdot k'$. The core idea of the reduction relies on transforming the decision of whether to select a vector into a solution S' for $\mathcal{I}$ into the decision of whether to resolve a tree gadget in G in one of two possible ways.

The set D consists of $(k' + 1)$ vertices $d_i^1, \ldots, d_i^{k'+1}$ for every i with $1 \leq i \leq k$. Moreover, for every $s \in S$ we introduce the gadget $G(s)$ defined as follows. $G(s)$ consists of $\max(s)$ stars with centers $c_1^s, \ldots, c_{\max(s)}^s$ and $d + 1$ leaves. For every star with center c_i^s, we denote by l_i^s one of its leaves (chosen arbitrarily). Additionally, $G(s)$ has a root vertex, denoted

by r^s, that has an edge to every center vertex c_i^s. Finally, we add edges between the leaves $l_1^s, \ldots, l_{\max(s)}^s$ and the vertices in D such that for every i and j with $1 \leq i \leq k$ and $1 \leq j \leq k'+1$, it holds that d_i^j has $s[i]$ neighbors among the leaves $l_1^s, \ldots, l_{\max(s)}^s$ of $G(s)$. Clearly this is always possible and can be done in an arbitrary manner.

We set d to be the maximum degree of the part of G constructed so far (note that this maximum is reached by one of the vertices in D). Moreover, we now ensure that for every i and j with $1 \leq i \leq k$ and $1 \leq j \leq k'+1$, the vertex d_i^j has degree $d + t[i]$ in G by attaching a appropriate number of leaves to d_i. Finally, we set ℓ to be $\left(\sum_{s \in S} \max(s)\right) + k'$. This completes the construction of $\mathcal{I}'$. Clearly, $\mathcal{I}'$ can be constructed in polynomial time. Moreover, $|D| \leq k \cdot k'$ and each component of $G - D$ is a tree with height at most 3. To complete the proof, it suffices to establish the equivalence between $\mathcal{I}$ and $\mathcal{I}'$. ◀

Clearly trees of height at most three are trivially acyclic. Moreover, it is easy to verify that such trees have pathwidth [25] and treedepth [32] at most three, which implies:

▶ **Corollary 9.** BDD *is* W[1]*-hard parameterized by any of the following parameters:*

- *the size of a feedback vertex set,*
- *the pathwidth and treedepth of the input graph,*
- *the size of a minimum set of vertices whose deletion results in components of pathwidth/treedepth at most three.*

4 Solving BDD using Treecut-width

The goal of this section is to provide a fixed-parameter algorithm for BDD parameterized by treecut-width. The core of the algorithm is a dynamic programming procedure which runs on a nice treecut decomposition of the input graph. First we define the data table the algorithm is going to dynamically compute for individual nodes of the treecut decomposition. For each node $t \in T$, the table is going to contain two components, which we will call the *universal cost* u_t and the *specific cost* s_t. Informally, the universal cost captures the minimum number of vertices which need to be deleted from Y_t to satisfy the degree bound in G_t. The specific cost captures how many more vertices (than the universal cost) we need to delete in order to satisfy the degree bound in G_t when we also place restrictions on how G_t will interact with the rest of the graph. We formalize these notions below.

Let us fix an instance (G, d, ℓ) of BDD and a treecut decomposition $(T, \mathcal{X})$ of G of width at most k and rooted at r. A *configuration* δ of a graph H with a designated vertex-subset Z is a mapping $Z \mapsto [k] \cup \text{del}$. Intuitively, configurations are going to be used to place additional restrictions on the deletion sets we are interested in. We let $\mathbf{bdd}(H, Z, \delta)$ denote the minimum size of a vertex set $W \subseteq V(H)$ such that:

- $v \in W \cap Z$ if and only if $\delta(v) = \text{del}$, and
- for each $v \in Z \setminus W$, the degree of v in $H - W$ is at most $d - \delta(v)$,
- for each $v \in V(H) \setminus (Z \cup W)$, the degree of v in $H - W$ is at most d.

Figure 2 depicts an illustration of $\mathbf{bdd}(H, Z, \delta)$. Informally, **bdd** captures the size of a minimum deletion set which intersects the designated subset precisely in the vertices specified by δ, and for the remainder of the designated subset it overshoots the degree bound by a buffer specified by δ. If $\mathbf{bdd}(H, Y, \delta)$ is not defined (which may happen, e.g., if $d < |Y|$), we formally set $\mathbf{bdd}(H, Y, \delta) = \infty$. For each node $t \in V(T)$, we can now define:

- $u_t = \mathbf{bdd}(G_t, \emptyset, \emptyset)$, and
- for each $\delta : \partial_t \to [k] \cup \text{del}$ such that each $v \in \partial_t$ is mapped to del or to an integer $i \leq |N(v) \setminus Y_t|$, we let $s'_t(\delta) = \mathbf{bdd}(G_t, \partial_t, \delta) - u_t$.

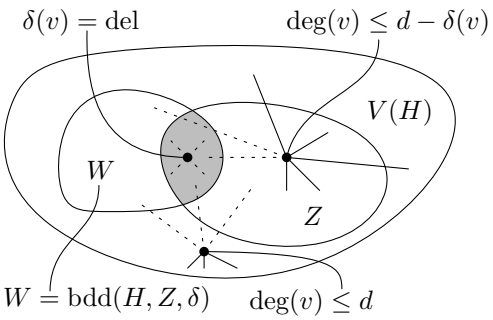


Figure 2 Illustration of the set $\mathbf{bdd}(H, Z, \delta)$. The dotted edges are not considered for the degree of a node v.

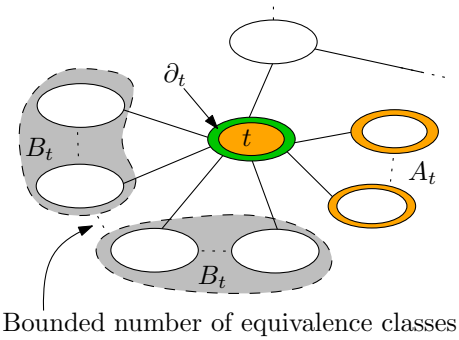


Figure 3 The three branching sets for a node $t \in T$, first branch on ∂_t (green), then on the boundaries of the bold nodes A_t together with the "interior" of t (orange) and finally on the equivalence classes of B_t (gray).

We proceed with a few observations. Naturally, the value of u_t can be much larger than k (as an example, consider a collection of disjoint stars), and this is not an issue for our algorithm. Furthermore, for every δ it holds that $0 \leq s'_t(\delta)$, since $u_t \leq \mathbf{bdd}(G_t, \partial_t, \delta)$; notice that u_t attains the value of the smallest deletion set for G_t, while $\mathbf{bdd}(G_t, \partial_t, \delta)$ attains the value of a smallest deletion set for G_t which satisfies certain additional restrictions.

Crucially, the value of $s'_t(\delta)$ can be much larger than k, and this represents a significant obstacle for our algorithm. The role of the specific cost in the dynamic programming procedure is to capture how a node may interact with the solution and how such interactions affect the size of a deletion set. The algorithm relies heavily on having only a bounded number of possible interactions in order to achieve its run-time bounds. Luckily, we will prove that any value of $s'_t(\delta)$ exceeding k must lead to a dead end and can be disregarded.

▶ **Lemma 10.** *Let S be a minimum-size bounded degree deletion set in G. Let δ^t_S be defined over ∂_t as follows: $\delta^t_S(v) = \text{del}$ if $v \in S$, and otherwise $\delta^t_S(v) = |(N(v) \setminus Y_t) \setminus S|$. Then $s'_t(\delta^t_S) \leq |N(Y_t)| \leq k$.*

Proof. For brevity, let $q = N(Y_t)$. The fact that $q \leq k$ follows immediately from the bound on the adhesion of t, hence we only need to prove that $s'_t(\delta^t_S) \leq q$. So, assume for a contradiction that $s'_t(\delta^t_S) > q$. Let P be a witness for the value of u_t, i.e., let P be a minimum-cardinality vertex subset of G_t such that the maximum degree in $G_t - P$ is at most d. Observe that $|P \cup N(Y_t)| = u_t + q$. Now consider the set $S' = (S \setminus Y_t) \cup P \cup N(Y_t)$. First of all, note that $|S'| < |S|$, since we obtained S' from S by removing more than $u_t + q$ vertices (recall that, by our assumption, $s'_t(\delta^t_S) > q$) and then adding back at most $u_t + q$ vertices. Second, we claim that S' is also a bounded degree deletion set in G. Indeed, consider for a contradiction that $G - S'$ contains a vertex v of degree higher than d. Such a v cannot lie in Y_t since P was a solution in G_t and $N(Y_t)$ separates G_t from the rest of G. On the other hand, v cannot lie outside of Y_t due to the fact that S itself was a solution in $G[V(G) - Y_t]$. So the claim holds, and S' contradicts the optimality of S. ◀

Thanks to Lemma 10, we can safely focus our attention on those configurations δ where $s'_t(\delta) \leq |N(Y_t)|$. Hence we define $s_t(\delta) = s'_t(\delta)$ if $s'_t(\delta) \leq |N(Y_t)|$ and $s'_t(\delta) = \infty$ otherwise. Observe that, unlike s'_t, the number of distinct possibilities of what a special cost s_t may look like is bounded by a function of k. The high-level strategy for the algorithm is now the following:

1. Compute (u_t, s_t) when t is a leaf,
2. Compute (u_t, s_t) when t is not a leaf, but the universal and specific costs are known for all of its children, and
3. Use the values (u_r, s_r) at the root node $r \in T$.

As we will see below, points **1.** and **3.** are straightforward.

▶ **Observation 11.** *(u_t, s_t) can be computed in time at most $2^{\mathcal{O}(k)}$ if t is a leaf.*

Proof. To compute u_t it suffices to exhaustively loop through all vertex subsets $L \subseteq X_t$ and check whether $G_t - L$ has degree at most d. Then u_t is equal to the minimum size of such a subset. To compute s_t, we proceed similarly: for each configuration δ such that each $v \in \partial_t$ is mapped to del or to an integer $i \leq |N(v) \setminus Y_t|$, we exhaustively loop through all $L \subseteq X_t \setminus \partial_t$ in order to determine the value of $\mathbf{bdd}(G_t, \partial_t, \delta)$, and we then use that value and u_t to determine $s_t(\delta)$. ◀

▶ **Observation 12.** *(G, d, ℓ) is a YES-instance of* BDD *if and only if $u_r \leq \ell$.*

Given the above, the last remaining obstacle is handling point (2), i.e., the dynamic propagation of information from leaves to the root. This is also the by far most challenging part of the algorithm. Let us fix a node $t \in T$ and for all its children p we assume (u_p, s_p) to be already computed.

Our strategy is to apply a 2-step approach. Figure 3 shows an illustration of the upcoming branching sets for a node t. Recall that A_t and B_t denote the set of all children of t which are bold and thin, respectively. First, we exhaustively loop over all options of how a deletion set candidate intersects with X_t and the borders of nodes in A_t, resulting in a set of "templates" which provide us with additional information about a potential solution. Here the bound on $|A_t|$ provided in Lemma 5 will be crucial. Second, we use branching and network flows to find an optimal way of extending such a template to a solution which deals with B_t. In this step, we overcome the fact that there may be an unbounded number of children p in B_t by "aggregating" them into types based on their s_p component. Lemma 10 along with our definition of specific costs then guarantees that the number of aggregated types will depend only on k. Informally, if two nodes p_1, p_2 in B_t have the same specific cost, then their behavior ("contribution") to any solution is fully interchangeable. In particular, even if p_1, p_2 have different universal costs, both of these costs will need to be "paid" by every solution regardless of how the solution handles the borders of these nodes. When formalizing the above sketched algorithm we obtain.

▶ **Lemma 13.** *Point* **2.** *can be solved in time $2^{\mathcal{O}(k^2)} \cdot |B_t|^2$, where $|B_t|$ is upper-bounded by the number of children of t.*

▶ **Theorem 14.** BDD *can be solved in time $n^3 + 2^{\mathcal{O}(k^2 \cdot \log k)} \cdot n^2$, where k and n are the treecut-width and number of vertices of the input graph, respectively.*

Proof. We begin by applying Theorem 3 followed by Lemma 4 to obtain a nice treecut decomposition $(T, \mathcal{X})$ of width at most $2k$. We then use a dynamic programming algorithm to compute the values u_t and s_t at every node $t \in T$. For leaves, this is carried out by Observation 11, while for non-leaves we invoke Lemma 13. Finally, once we compute u_r for the root r, we can determine the answer to a BDD instance using Observation 12. ◀

Finally, using standard techniques it is not difficult to show that BDD parameterized by treecut width does not admit a polynomial kernel.

▶ **Theorem 15.** *BDD parameterized by treecut width has no polynomial kernel unless* coNP $\subseteq$ NP/poly.

5 Core Fracture Number

In this section we introduce the new structural parameter core fracture number and provide a fixed-parameter algorithm for BDD parameterized by this parameter. An important prerequisite for the introduction of this parameter is the following simple preprocessing procedure that can be applied to any BDD instance. Given an instance $\mathcal{I} = (G, d, \ell)$ of BDD, the *core* of $\mathcal{I}$, denoted by $\mathbf{c}(\mathcal{I}) = (\mathbf{c}(G), d, \ell)$, is the BDD instance obtained from $\mathcal{I}$ after removing all edges whose both endpoints have degree at most d from G.

▶ **Observation 16.** *Let $\mathcal{I} = (G, d, \ell)$ be a BDD instance. Then $\mathcal{I}$ and $\mathbf{c}(\mathcal{I})$ are equivalent instances of BDD in the sense that any solution for $\mathcal{I}$ is also a solution for $\mathbf{c}(\mathcal{I})$ and vice versa. Moreover, $\mathbf{c}(\mathcal{I})$ can be computed in linear time w.r.t. the number of edges of G.*

In the following we will assume that we have already applied the above preprocessing procedure to any BDD instance and hence the graph of the instance does not contain any edges between vertices whose degree is already below the given degree bound. The *core fracture number* of a BDD instance $\mathcal{I} = (G, d, \ell)$, denoted by $\mathbf{cfn}(\mathcal{I})$, is the minimum integer k such that there is a deletion set $D \subseteq V(G)$ with $|D| \leq k$ and the number of vertices in any component C of $G \setminus D$ of degree larger than d in G is at most k. In other words, each connected component of $G - D$ may contain only at most k vertices of degree greater than d. We start by showing that this parameter is orthogonal to treecut width.

▶ **Theorem 17.** *There is a class of BDD instances with bounded treecut width and unbounded core fracture number. Similarly, there is a class of BDD instances with bounded core fracture number and unbounded treecut width. Moreover, both classes only contain BDD instances $\mathcal{I}$ with $\mathbf{c}(\mathcal{I}) = \mathcal{I}$.*

We are now ready to present our fixed-parameter algorithm for BDD parameterized by the core fracture number. The algorithm consists of two steps: (1) it computes a deletion set D of size at most k, witnessing that $\mathbf{cfn}(\mathcal{I}) \leq k$ and (2) it solves $\mathcal{I}$ with the help of the deletion set D. Namely, our algorithm will consists of fixed-parameter algorithms for the following two parameterized problems. Given an instance $\mathcal{I} = (G, d, \ell)$ of BDD and an integer k (which also serves as the parameter of the problem), the Core Fracture Number Detection (CFND) problem, asks whether $\mathbf{cfn}(\mathcal{I}) \leq k$ and if so outputs a set $D \subseteq V(G)$ witnessing this. On the other hand the Core Fracture Number Evaluation (CFNE) problem asks whether $\mathcal{I}$ has a solution for a given BDD instance $\mathcal{I} = (G, d, \ell)$ and a set $D \subseteq V(G)$ witnessing that $\mathbf{cfn}(\mathcal{I}) \leq |D|$ and is parameterized by $|D|$.

▶ **Theorem 18.** *CFND can be solved in time $\mathcal{O}((2k+1)^k|E(G)|)$ and is hence fixed-parameter tractable. Moreover, CFND can be approximated in polynomial time within a factor of $2k+1$.*

Let $\mathcal{I} = (G, d, \ell, D)$ be an instance of CFNE and assume w.l.o.g. that $\mathbf{c}(G) = G$. We start by showing that we do not need to consider solutions $V' \subseteq V(G)$ for $\mathcal{I}$ that contain more than $2k - 1$ vertices from any component C of $G - D$.

▶ **Lemma 19.** *If $\mathcal{I}$ has a solution, then it has a solution V' such that $|V' \cap V(C)| < 2k$ for every component C of $G - D$.*

Proof. Let V' be a solution for $\mathcal{I}$ and C be a component of $G - D$ with $|V' \cap V(C)| \geq 2k$; if no such component exists, then we are done. Let M be the set of all vertices in C, whose degree is larger than d in G. Then $(V' \setminus V(C)) \cup M \cup D$ is also a solution for $\mathcal{I}$ and moreover $|(V' \setminus V(C)) \cup M \cup D| \leq |V'| - 2k + k + k \leq |V'|$. By iterating the same process for every component C with $|V' \cap V(C)| \geq 2k$, one obtains the desired solution for $\mathcal{I}$. ◀

Let C be a component of $G - D$ and let $M \subseteq V(C)$ be the set of all vertices with degree larger than d in G. Then the *signature* of C, denoted by $\mathcal{S}(C)$, contains all pairs (D', Γ) such that $D' \subseteq D$, and Γ is the set of all pairs (o, γ) such that:

- o is an integer with $0 \leq o < 2k$, and
- $\gamma : D \setminus D' \to \{0, \ldots, 2k-1\}$ is a mapping such that there is a set $V' \subseteq V(C)$ with $|V'| = o$ satisfying the following conditions:
  - **(S1)** every vertex in $M \setminus V'$ has degree at most d in $G - (V' \cup D')$ and
  - **(S2)** for every vertex v in $D \setminus D'$, V' contains exactly $\gamma(v)$ neighbors of v.

Informally, for every subset D' of vertices that we decide to delete from D, the signature tells us how many vertices in C we need to delete and how their deletion affects the degrees of the remaining vertices in $D - D'$. Because we only need to consider solutions containing less than $2k$ vertices from C (Lemma 19), the number of ways in which different solutions effect the degrees of vertices in D is bounded, which allows us to compute the signatures.

▶ **Lemma 20.** *The signature $\mathcal{S}(C)$ can be computed in time $O(|V(C)| + |E(C)| + 2^k(2k)^{2^{2k}})$ for any component C of $G - D$.*

Let $D' \subseteq D$ and C and C' be two distinct components of $G - D$. We say that C and C' are *equivalent w.r.t.* D' if $(D', \Gamma) \in \mathcal{S}(C) \cap \mathcal{S}(C')$ for some Γ. Let $\mathcal{P}(D')$ be the partition of all components of $G - D$ into equivalence classes and for an equivalence class $\mathcal{C} \in \mathcal{P}(D')$ let $\Gamma(\mathcal{C})$ denote the set Γ such that $(D', \Gamma) \in \mathcal{S}(C)$ for every $C \in \mathcal{C}$. Note that $|\mathcal{P}(D')| \leq 2^{k(2k)^k}$.

▶ **Lemma 21.** *An instance $\mathcal{I} = (G, d, \ell, D)$ has a solution if and only if there is a subset D' of D and a mapping α that assigns to every $\mathcal{C} \in \mathcal{P}(D')$ and every $(o, \gamma) \in \Gamma(\mathcal{C})$ a natural number satisfying the following conditions:*

(C1) $(\sum_{\mathcal{C} \in \mathcal{P}(D') \wedge (o,\gamma) \in \Gamma(\mathcal{C})} o \cdot \alpha(\mathcal{C}, (o, \gamma))) + |D'| \leq \ell$, *i.e., the budget ℓ is not exceeded,*
(C2) $\sum_{(o,\gamma) \in \Gamma(\mathcal{C})} \alpha(\mathcal{C}, (o, \gamma)) = |\mathcal{C}|$ *for every $\mathcal{C} \in \mathcal{P}(D')$, i.e., all components are considered,*
(C3) $\sum_{\mathcal{C} \in \mathcal{P}(D') \wedge (o,\gamma) \in \Gamma(\mathcal{C})} \gamma(v) \cdot \alpha(\mathcal{C}, (o, \gamma) \geq |N_{G-D'}(v)| - d$ *for every $v \in D \setminus D'$, i.e., the degree conditions for the vertices in $D \setminus D'$ are satisfied.*

Proof. Towards showing the forward direction let V' be a solution for $\mathcal{I}$. We start by setting $D' = D \cap V'$. Consider a component C of $G - D$ and let Γ be the set such that $(D', \Gamma) \in \mathcal{S}(C)$. Because of Lemma 19, we can assume that $|V' \cap V(C)| < 2k$. Hence Γ contains a pair $(|V' \cap V(C)|, \gamma)$, which we denote by $A(C)$, such that for every $v \in D \setminus D'$, it holds that v has exactly $\gamma(v)$ neighbors in $V' \cap V(C)$. For every $\mathcal{C} \in \mathcal{P}(D')$ and $(o, \gamma) \in \Gamma(\mathcal{C})$, we now set $\alpha(\mathcal{C}, (o, \gamma))$ to be the number of components C in $\mathcal{C}$ with $A(C) = (o, \gamma)$ and claim that α satisfies the conditions (C1)–(C3). Because $(\sum_{\mathcal{C} \in \mathcal{P}(D') \wedge (o,\gamma) \in \Gamma(\mathcal{C})} o \cdot \alpha(\mathcal{C}, (o, \gamma))) + |D'| = |V'|$ and $|V'| \leq \ell$, we obtain that α satisfies (C1). Condition (C2) follows immediately from the definition of α. Finally, Condition (C3) follows because for every $v \in D \setminus D'$ it holds that $\sum_{\mathcal{C} \in \mathcal{P}(D') \wedge (o,\gamma) \in \Gamma(\mathcal{C})} \gamma(v) \cdot \alpha(\mathcal{C}, (o, \gamma))$ is equal to the number of neighbors of v in $V' \setminus D$ and the fact that v can have at most d neighbors in $G - V'$.

Towards showing the reverse direction let $D' \subseteq D$ and α be a mapping satisfying (C1)–(C3). For a component $C \in \mathcal{C}$ and $(o, \gamma) \in \Gamma$, where $\mathcal{C} \in \mathcal{P}(D')$ and $(D', \Gamma) \in \mathcal{S}(C)$, we denote by $V(C, (o, \gamma))$ a subset of $V(C)$ of size o satisfying the conditions (S1) and (S2) in the definition of a signature. Then a solution V' for $\mathcal{I}$ is obtained as follows. For any $\mathcal{C} \in \mathcal{P}(D')$ we take the union of $V(C, (o, \gamma))$ for exactly $\alpha(\mathcal{C}, (o, \gamma))$ components $C \in \mathcal{C}$. Condition (C2) ensures that there are enough components in $\mathcal{C}$ and moreover that this way we use every component exactly once. Finally, we add D' to V'. Because of Condition (C1), we have that $|V'| \leq \ell$. Moreover, because of Condition (C3), we obtain that every vertex in $D \setminus D'$ has degree at most d in $G - V'$. The same holds for every vertex in any component C of $G - D$, because of Property (S1). Hence V' is a solution for $\mathcal{I}$ of size at most ℓ. ◀

With the help of the above lemma, we can express the existence of a solution in terms of the solution of an integer linear program with a bounded number of variables, which in turn can be solved in fpt-time w.r.t. the number of variables [28].

▶ **Theorem 22.** *CFNE is fixed-parameter tractable.*

Proof sketch. Let $\mathcal{I} = (G, d, \ell, D)$ be the given instance of CFNE. The algorithm first computes the signature $\mathcal{S}(C)$ for every component C of $G - D$ according to Lemma 20. It then uses the characterization given in Lemma 21 to decide whether $\mathcal{I}$ has a solution. Namely, for every $D' \subseteq D$ the algorithm constructs an ILP instance $\mathcal{I}'$ whose optimum is at most $\ell - |D'|$ if and only if the BDD instance $\mathcal{I}$ has a solution V' with $V' \cap D = D'$. In accordance with Lemma 21 the ILP instance $\mathcal{I}'$ has one variable, denote by $x_{\mathcal{C},(o,\gamma)}$, for every $\mathcal{C} \in \mathcal{P}(D')$ and $(o, \gamma) \in \Gamma(\mathcal{C})$ and consists of the following constraints:

$$\begin{aligned} \text{minimize} \quad & \sum_{\mathcal{C}\in\mathcal{P}(D'),(o,\gamma)\in\Gamma(\mathcal{C})} o \cdot x_{\mathcal{C},(o,\gamma)} \\ \text{subject to} \quad & \sum_{(o,\gamma)\in\Gamma(\mathcal{C})} x_{\mathcal{C},(o,\gamma)} = |\mathcal{C}| && \forall \mathcal{C} \in \mathcal{P}(D') \\ & \sum_{\mathcal{C}\in\mathcal{P}(D')\wedge(o,\gamma)\in\Gamma(\mathcal{C})} \gamma(v) \cdot x_{\mathcal{C},(o,\gamma)} \geq |N_{G-D'}(v)| - d && \forall v \in D \setminus D' \end{aligned}$$

Observe that there is a one-to-one correspondence between assignments β for the variables in $\mathcal{I}'$ and the assignment α defined in Lemma 21. Moreover, the constraints of $\mathcal{I}'$ ensure Condition (C2) and (C3) and Condition (C1) can be satisfied if and only if the optimum value of $\mathcal{I}'$ is at most $\ell - |D'|$. This completes the description of the algorithm. ◀

As our final result, we show a kernel lower bound for CFNE.

▶ **Theorem 23.** *CFNE has no polynomial kernel unless* coNP $\subseteq$ NP/poly.

Proof sketch. We give a *polynomial parameter transformation* [2, Proposition 1] from the well-known Set Cover problem parameterized by the size of the universe. It is known that Set Cover does not admit a polynomial kernel unless coNP $\subseteq$ NP/poly [10]. Given an instance $\mathcal{I} = (U, \mathcal{F}, k)$ of Set Cover, we construct an instance $\mathcal{I}' = (G, d, \ell, D)$ of CFNE as follows. G has one vertex v_u for every $u \in U$ as well as one vertex w_F for every $F \in \mathcal{F}$. Moreover, G has an edge between a vertex v_u and a vertex w_F if and only if $u \in F$. We set $D = \{ v_u \mid u \in U \}$. Let Δ be the maximum degree of any vertex in G. Then we attach to every vertex in D new leaf vertices such that the degree of every vertex in D becomes $\Delta + 1$. This completes the construction of G. Finally, we set $d = \Delta$ and $\ell = k$. Because $G - D$ is an independent set, this shows that $\mathbf{cfn}(G) \leq k$. To complete the proof, it remains to show that $\mathcal{I}$ has a solution if and only if so does $\mathcal{I}'$. ◀

6 Concluding Notes

Our results close a wide gap in the understanding of the complexity landscape of BDD parameterized by structural parameters. In particular, they not only resolve the main open question from previous work in the area [5], but push the lower bounds significantly further, specifically to deletion distance to trees of bounded depth. Moreover, we identified structural parameterizations which are better suited for the problem at hand and used these to obtain two novel fixed-parameter algorithms for BDD.

References

1 Martin Aigner and Günter M. Ziegler. *Proofs from the Book (3. ed.)*. Springer, 2004.
2 Christer Bäckström, Peter Jonsson, Sebastian Ordyniak, and Stefan Szeider. A complete parameterized complexity analysis of bounded planning. *JCSS*, 81(7):1311–1332, 2015.
3 Balabhaskar Balasundaram, Sergiy Butenko, and Illya V. Hicks. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research*, 59(1):133–142, 2011.
4 Balabhaskar Balasundaram, Shyam Sundar Chandramouli, and Svyatoslav Trukhanov. Approximation algorithms for finding and partitioning unit-disk graphs into co-k-plexes. *Optimization Letters*, 4(3):311–320, 2010.
5 Nadja Betzler, Robert Bredereck, Rolf Niedermeier, and Johannes Uhlmann. On bounded-degree vertex deletion parameterized by treewidth. *Discrete Applied Mathematics*, 160(1-2):53–60, 2012.
6 Hans L. Bodlaender and Babette van Antwerpen-de Fluiter. Reduction algorithms for graphs of small treewidth. *Inf. Comput.*, 167(2):86–119, 2001.
7 Zhi-Zhong Chen, Michael R. Fellows, Bin Fu, Haitao Jiang, Yang Liu, Lusheng Wang, and Binhai Zhu. A linear kernel for co-path/cycle packing. In *Proc. AAIM 2010*, volume 6124 of *LNCS*, pages 90–102. Springer, 2010.
8 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.
9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
11 Anders Dessmark, Klaus Jansen, and Andrzej Lingas. The maximum k-dependent and f-dependent set problem. In *Proc. ISAAC 1993*, volume 762 of *LNCS*, pages 88–98. Springer, 1993.
12 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer Verlag, New York, 2nd edition, 2000.
13 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
14 Eduard Eiben, Robert Ganian, and Stefan Szeider. Meta-kernelization using well-structured modulators. In Thore Husfeldt and Iyad A. Kanj, editors, *Proc. IPEC 2015*, volume 43 of *LIPIcs*, pages 114–126. Leibniz-Zentrum für Informatik, 2015.
15 Eduard Eiben, Robert Ganian, and Stefan Szeider. Solving problems on graphs of high rank-width. In *Proc. WADS 2015*, volume 9214 of *LNCS*, pages 314–326. Springer, 2015.
16 Paul Erdős and Paul Turán. On a problem of Sidon in additive number theory, and on some related problems. *Journal of the London Mathematical Society*, 1(4):212–215, 1941.
17 Michael R. Fellows, Jiong Guo, Hannes Moser, and Rolf Niedermeier. A generalization of Nemhauser and Trotter's local optimization theorem. *J. Comput. Syst. Sci.*, 77(6):1141–1158, 2011.
18 Arnaud Fréville. The multidimensional 0-1 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1–21, 2004.
19 Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. *J. Comput. Syst. Sci.*, 84:219–242, 2017.
20 Robert Ganian, Eun Jung Kim, and Stefan Szeider. Algorithmic applications of tree-cut width. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Proc. MFCS 2015*, volume 9235 of *LNCS*, pages 348–360. Springer, 2015.

21 Robert Ganian, Friedrich Slivovsky, and Stefan Szeider. Meta-kernelization with structural parameters. *J. Comput. Syst. Sci.*, 82(2):333–346, 2016.

22 Serge Gaspers, Neeldhara Misra, Sebastian Ordyniak, Stefan Szeider, and Stanislav Zivny. Backdoors into heterogeneous classes of SAT and CSP. *J. Comput. Syst. Sci.*, 85:38–56, 2017.

23 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

24 Eunjung Kim, Sang-il Oum, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. An FPT 2-approximation for tree-cut decomposition. In Laura Sanità and Martin Skutella, editors, *Proc. WAOA 2015*, volume 9499 of *LNCS*, pages 35–46. Springer, 2015.

25 Ton Kloks. *Treewidth: Computations and Approximations*, volume 842 of *LNCS*. Springer Verlag, Berlin, 1994.

26 Christian Komusiewicz, Falk Hüffner, Hannes Moser, and Rolf Niedermeier. Isolation concepts for efficiently enumerating dense subgraphs. *Theor. Comput. Sci.*, 410(38-40):3640–3654, 2009.

27 Martin Kronegger, Sebastian Ordyniak, and Andreas Pfandler. Variable-deletion backdoors to planning. In *Proc. AAAI 2015*, pages 2300–2307. AAAI Press, 2014.

28 H. W. Lenstra. Integer programming with a fixed number of variables. *MATH. OPER. RES*, 8(4):538–548, 1983.

29 Dániel Marx and Paul Wollan. Immersions in highly edge connected graphs. *SIAM J. Discrete Math.*, 28(1):503–520, 2014.

30 Benjamin McClosky and Illya V. Hicks. Combinatorial algorithms for the maximum k-plex problem. *J. Comb. Optim.*, 23(1):29–49, 2012.

31 Hannes Moser, Rolf Niedermeier, and Manuel Sorge. Exact combinatorial algorithms and experiments for finding maximum k-plexes. *J. Comb. Optim.*, 24(3):347–373, 2012.

32 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.

33 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

34 Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. Fast fixed-parameter tractable algorithms for nontrivial generalizations of vertex cover. *Discrete Applied Mathematics*, 152(1-3):229–245, 2005.

35 Stephen B Seidman and Brian L Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology*, 6(1):139–154, 1978.

36 Paul Wollan. The structure of graphs not admitting a fixed immersion. *J. Comb. Theory, Ser. B*, 110:47–66, 2015.

Dependences in Strategy Logic

Patrick Gardy
LSV, CNRS & ENS Paris-Saclay, Univ. Paris-Saclay, France

Patricia Bouyer
LSV, CNRS & ENS Paris-Saclay, Univ. Paris-Saclay, France

Nicolas Markey
LSV, CNRS & ENS Paris-Saclay, Univ. Paris-Saclay, France and
Univ. Rennes, CNRS, Inria, IRISA, France

Abstract

Strategy Logic (SL) is a very expressive temporal logic for specifying and verifying properties of multi-agent systems: in SL, one can quantify over strategies, assign them to agents, and express LTL properties of the resulting plays. Such a powerful framework has two drawbacks: first, model checking SL has non-elementary complexity; second, the exact semantics of SL is rather intricate, and may not correspond to what is expected. In this paper, we focus on *strategy dependences* in SL, by tracking how existentially-quantified strategies in a formula may (or may not) depend on other strategies selected in the formula, revisiting the approach of [Mogavero *et al.*, Reasoning about strategies: On the model-checking problem, 2014]. We explain why *elementary* dependences, as defined by Mogavero *et al.*, do not exactly capture the intended concept of behavioral strategies. We address this discrepancy by introducing *timeline* dependences, and exhibit a large fragment of SL for which model checking can be performed in *2*-EXPTIME under this new semantics.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics, Theory of computation → Verification by model checking

Keywords and phrases Strategic Reasoning, Strategy Logic, Dependences, Behavioural Strategies

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.34

Funding Supported by ERC project EQualIS (308087).

1 Introduction

Temporal logics. Since Pnueli's seminal paper [24] in 1977, temporal logics have been widely used in theoretical computer science, especially by the formal-verification community. Temporal logics provide powerful languages for expressing properties of reactive systems, and enjoy efficient algorithms for satisfiability and model checking [9]. Since the early 2000s, new temporal logics have appeared to address *open* and *multi-agent systems*. While classical temporal logics (e.g. CTL [8, 25] and LTL [24]) could only deal with one or all the behaviours of the whole system, ATL [2] expresses properties of (executions generated by) behaviours of individual components of the system. ATL has been extensively studied since then, both about its expressiveness and about its verification algorithms [2, 13, 16].

Strategic interactions in ATL. Strategies in ATL are handled in a very limited way, and there are no real *strategic interactions* in that logic (which, in return, enjoys a polynomial-time model-checking algorithm). Over the last 10 years, various extensions have been defined and studied in order to allow for more interactions [1, 7, 6, 18, 26]. *Strategy Logic* (SL for short) [7, 18] is such a powerful approach, in which strategies are first-class objects; formulas

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 34; pp. 34:1–34:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


can quantify (universally and existentially) over strategies, store those strategies in variables, assign them to players, and express properties of the resulting plays. As a simple example, the existence of a winning strategy for Player A (with objective φ_A) against any strategy of Player B would be written as $\exists \sigma_A.\ \forall \sigma_B.\ \mathsf{assign}(A \mapsto \sigma_A; B \mapsto \sigma_B).\ \varphi_A$. This makes the logic both expressive and easy to use (at first sight), at the expense of a very high complexity: SL model checking has non-elementary complexity, and satisfiability is undecidable [18, 15].

Understanding SL. Since it enjoys decidable model checking and high expressiveness, SL is the logic of choice for showing that some game problems are decidable (e.g. rational synthesis [12, 14, 10] or assume-admissible synthesis [5]). For instance, the existence of an admissible strategy for player A (i.e., a strategy that is strictly dominated by no other strategies [5]) is expressed as

$$\exists \sigma_A.\ \forall \sigma'_A.\ \left[\begin{array}{l} \exists \sigma_B.\ (\mathsf{assign}(A \mapsto \sigma_A, B \mapsto \sigma_B).\varphi_A \wedge \mathsf{assign}(A \mapsto \sigma'_A, B \mapsto \sigma_B).\neg\varphi_A) \\ \vee \\ \forall \sigma'_B.\ (\mathsf{assign}(A \mapsto \sigma_A, B \mapsto \sigma'_B).\varphi_A \vee \mathsf{assign}(A \mapsto \sigma'_A, B \mapsto \sigma'_B).\neg\varphi_A) \end{array}\right]$$

However, it has been noticed in recent works that the nice expressiveness of SL comes with unexpected phenomena. One such phenomenon is induced by the separation of strategy quantification and strategy assignment: are the events between strategy quantifications and strategy assignments part of the *memory* of the strategy? While both options may make sense depending on the applications, only one of them makes model checking decidable [4].

A second phenomenon—which is the main focus of the present paper—concerns *strategy dependences* [18]: in a formula such as $\forall \sigma_A.\ \exists \sigma_B.\ \xi$, the existentially-quantified strategy σ_B may depend on *the whole* strategy σ_A; in other terms, the action returned by strategy σ_B after some finite history ρ may depend on what strategy σ_A would play on any other history ρ'. Again, in some contexts, it may be desirable that the value of strategy σ_B after history ρ can be *computed* based solely on what has been observed along ρ (see Fig. 2 for an illustration). This approach was initiated in [18, 21], conjecturing that large fragments of SL (subsuming ATL *) would have *2*-EXPTIME model-checking algorithms with such limited dependences.

Our contributions. We follow this line of work by performing a more thorough exploration of strategy dependences in (a fragment of) SL. We mainly follow the framework of [21], based on a kind of Skolemization of the formula: for instance, a formula of the form $(\forall x_i \exists y_i)_i.\ \xi$ is satisfied if there exists a *dependence map* θ defining each existentially-quantified strategy y_j based on the universally-quantified strategies $(x_i)_i$. In order to recover the classical semantics of SL, it is only required that the strategy $\theta((x_i)_i)(y_j)$ (i.e. the strategy assigned to the existentially-quantified variable y_j by $\theta((x_i)_i)$) only depends on $(x_i)_{i<j}$.

Based on this definition, other constraints can be imposed on dependence maps, in order to refine the dependences of existentially-quantified strategies on universally-quantified ones. *Elementary dependences* [21] only allows existentially-quantified strategy y_j to depend on the values of $(x_i)_{i<j}$ along the current history. This gives rise to two different semantics in general, but fragments of SL have been defined (SL[1G] in [17], SL[CG] and SL[DG] in [20]) on which the classic and elementary semantics would coincide.

The coincidence actually only holds for SL[1G]. As we explain in this paper, elementary dependences as defined and used in [17, 20] do not exactly capture the intuition that strategies should depend on the "behavior [of universal strategies] on the history of interest only" [20]: indeed, they only allow dependences on universally-quantified strategies *that appear earlier in the formula*, while we claim that the behaviour of all universally-quantified

strategies should be considered. We address this discrepancy by introducing another kind of dependences, which we call *timeline dependences*, and which extend elementary dependences by allowing existentially-quantified strategies to additionally depend on *all* universally-quantified strategies along *strict prefixes* of the current history (as illustrated on Fig. 4).

We study and compare those three dependences (classic, elementary and timeline), showing that they correspond to three distinct semantics. Because the semantics based on dependence maps is defined in terms of the *existence* of a witness map, we show that the syntactic negation of a formula may not correspond to its semantic negation: there are cases where both a formula φ and its syntactic negation $\neg\varphi$ fail to hold (i.e., none of them has a witness map). This phenomenon is already present, but had not been formally identified, in [18, 21]. The main contribution of the present paper is the definition of a large (and, in a sense, maximal) fragment of SL for which syntactic and semantic negations coincide under the timeline semantics. As an (important) side result, we show that model checking this fragment under the timeline semantics is *2*-EXPTIME-complete.

2 Definitions

2.1 Concurrent game structures

Let AP be a set of atomic propositions, $\mathcal{V}$ be a set of variables, and Agt be a set of agents. A *concurrent game structure* is a tuple $\mathcal{G} = (\mathsf{Act}, \mathsf{Q}, \Delta, \mathsf{lab})$ where Act is a finite set of actions, Q is a finite set of states, $\Delta\colon \mathsf{Q} \times \mathsf{Act}^{\mathsf{Agt}} \to \mathsf{Q}$ is the transition function, and $\mathsf{lab}\colon \mathsf{Q} \to 2^{\mathsf{AP}}$ is a labelling function. An element of $\mathsf{Act}^{\mathsf{Agt}}$ will be called a *move vector*. For any $q \in \mathsf{Q}$, we let $\mathsf{succ}(q)$ be the set $\{q' \in Q \mid \exists m \in \mathsf{Act}^{\mathsf{Agt}}.\ q' = \Delta(q, m)\}$. For the sake of simplicity, we assume in the sequel that $\mathsf{succ}(q) \neq \varnothing$ for any $q \in Q$. A game $\mathcal{G}$ is said *turn-based* whenever for every state $q \in \mathsf{Q}$, there is a player $\mathsf{own}(()q) \in \mathsf{Agt}$ (named the *owner* of q) such that for any two move vectors m_1 and m_2 with $m_1(\mathsf{own}(()q)) = m_2(\mathsf{own}(()q))$, it holds $\Delta(q, m_1) = \Delta(q, m_2)$. Figure 1 displays an example of a (turn-based) game.

Fix a state $q \in \mathsf{Q}$. A *play* in $\mathcal{G}$ from q is an infinite sequence $\pi = (q_i)_{i\in\mathbb{N}}$ of states in Q such that $q_0 = q$ and $q_i \in \mathsf{succ}(q_{i-1})$ for all $i > 0$. We write $\mathsf{Play}_{\mathcal{G}}(q)$ for the set of plays in $\mathcal{G}$ from q. In this and all similar notations, we might omit to mention $\mathcal{G}$ when it is clear from the context, and q when we consider the union over all $q \in Q$. A (strict) prefix of a play π is a finite sequence $\rho = (q_i)_{0\leq i\leq L}$, for some $L \in \mathbb{N}$. We write $\mathsf{Pref}(\pi)$ for the set of strict prefixes of play π. Such finite prefixes are called *histories*, and we let $\mathsf{Hist}_{\mathcal{G}}(q) = \mathsf{Pref}(\mathsf{Play}_{\mathcal{G}}(q))$. We extend the notion of strict prefixes and the notation Pref to histories in the natural way, requiring in particular that $\rho \notin \mathsf{Pref}(\rho)$. A (finite) extension of a history ρ is any history ρ' such that $\rho \in \mathsf{Pref}(\rho')$. Let $\rho = (q_i)_{i\leq L}$ be a history. We define $\mathsf{first}(\rho) = q_0$ and $\mathsf{last}(\rho) = q_L$. Let $\rho' = (q'_j)_{j\leq L'}$ be a history from $\mathsf{last}(\rho)$. The *concatenation* of ρ and ρ' is then defined as the path $\rho \cdot \rho' = (q''_k)_{k\leq L+L'}$ such that $q''_k = q_k$ when $k \leq L$ and $q''_k = q'_{k-L}$ when $L \geq k$ (notice that we required $q'_0 = q_L$).

A *strategy* from q is a mapping $\delta\colon \mathsf{Hist}_{\mathcal{G}}(q) \to \mathsf{Act}$. We write $\mathsf{Strat}_{\mathcal{G}}(q)$ for the set of strategies in $\mathcal{G}$ from q. Given a strategy $\delta \in \mathsf{Strat}(q)$ and a history ρ from q, the *translation* $\delta_{\overrightarrow{\rho}}$ of δ by ρ is the strategy $\delta_{\overrightarrow{\rho}}$ from $\mathsf{last}(\rho)$ defined by $\delta_{\overrightarrow{\rho}}(\rho') = \delta(\rho \cdot \rho')$ for any $\rho' \in \mathsf{Hist}(\mathsf{last}(\rho))$. A *valuation* from q is a partial function $\chi\colon \mathcal{V} \cup \mathsf{Agt} \rightharpoonup \mathsf{Strat}(q)$. As usual, for any partial function f, we write $\mathsf{dom}(f)$ for the domain of f.

Let $q \in Q$ and χ be a valuation from q. If $\mathsf{Agt} \subseteq \mathsf{dom}(\chi)$, then χ induces a unique play from q, called its *outcome*, and defined as $\mathsf{out}(q, \chi) = (q_i)_{i\in\mathbb{N}}$ such that $q_0 = q$ and for every $i \in \mathbb{N}$, we have $q_{i+1} = \Delta(q_i, m_i)$ with $m_i(A) = \chi(A)((q_j)_{j\leq i})$ for every $A \in \mathsf{Agt}$.

2.2 Strategy Logic with boolean goals

Strategy Logic (SL for short) was introduced in [7], and further extended and studied in [22, 18], as a rich logical formalism for expressing properties of games. SL manipulates strategies as first-order elements, assigns them to players, and expresses LTL properties on the outcomes of the resulting strategic interactions. This results in a very expressive temporal logic, for which satisfiability is undecidable [22, 19] and model checking is TOWER-complete [18, 3]. In this paper, we focus on a restricted fragment of SL, called SL[BG]$^\flat$ (where BG stands for *boolean goals* [18], and the symbol $\flat$ indicates that we do not allow nesting of (closed) subformulas; we discuss this latter restriction below).

Syntax. Formulas in SL[BG]$^\flat$ are built along the following grammar

$$\mathsf{SL[BG]}^\flat \ni \varphi ::= \exists x.\ \varphi \mid \forall x.\ \varphi \mid \xi \qquad \xi ::= \neg\xi \mid \xi \wedge \xi \mid \xi \vee \xi \mid \beta$$
$$\beta ::= \mathsf{assign}(\sigma).\ \psi \qquad \psi ::= \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \mathbf{X}\,\psi \mid \psi\,\mathbf{U}\,\psi \mid p$$

where x ranges over $\mathcal{V}$, σ ranges over the set $\mathcal{V}^{\mathsf{Agt}}$ of *full assignments*, and p ranges over AP. A *goal* is a formula of the form β in the grammar above; it expresses an LTL property ψ on the outcome of the mapping σ. Formulas in SL[BG]$^\flat$ are thus made of an initial block of first-order quantifiers (selecting strategies for variables in $\mathcal{V}$), followed by a boolean combination of goals.

Free variables. With any subformula ζ of some formula $\varphi \in \mathsf{SL[BG]}^\flat$, we associate its set of *free agents and variables*, which we write $\mathsf{free}(\zeta)$. It contains the agents and variables that have to be associated with a strategy in order to unequivocally evaluate ζ (as will be seen from the definition of the semantics of SL[BG]$^\flat$ below). The set $\mathsf{free}(\zeta)$ is defined inductively:

$$\mathsf{free}(p) = \varnothing \quad \text{for all } p \in \mathsf{AP} \qquad \mathsf{free}(\mathbf{X}\,\psi) = \mathsf{Agt} \cup \mathsf{free}(\psi)$$
$$\mathsf{free}(\neg\alpha) = \mathsf{free}(\alpha) \qquad \mathsf{free}(\psi_1\,\mathbf{U}\,\psi_2) = \mathsf{Agt} \cup \mathsf{free}(\psi_1) \cup \mathsf{free}(\psi_2)$$
$$\mathsf{free}(\alpha_1 \vee \alpha_2) = \mathsf{free}(\alpha_1) \cup \mathsf{free}(\alpha_2) \qquad \mathsf{free}(\exists x.\ \varphi) = \mathsf{free}(\varphi) \setminus \{x\}$$
$$\mathsf{free}(\alpha_1 \wedge \alpha_2) = \mathsf{free}(\alpha_1) \cup \mathsf{free}(\alpha_2) \qquad \mathsf{free}(\forall x.\ \varphi) = \mathsf{free}(\varphi) \setminus \{x\}$$
$$\mathsf{free}(\mathsf{assign}(\sigma).\ \varphi) = (\mathsf{free}(\varphi) \cup \sigma(\mathsf{Agt} \cap \mathsf{free}(\varphi))) \setminus \mathsf{Agt}$$

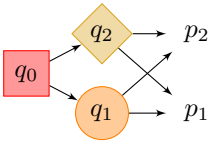
Subformula ζ is said to be *closed* whenever $\mathsf{free}(\zeta) = \varnothing$. We can now comment on our choice of considering the flat fragment of SL[BG]: the full fragment, as defined in [18], allows for nesting *closed* SL[BG] formulas in place of atomic propositions. The meaning of such nesting in our setting is ambiguous, because our semantics (in Sections 3 to 5) are defined in terms of the *existence of a witness*, which does not easily propagate in formulas. In particular, as we explain later in the paper, the semantics of the negation of a formula (there is a witness for $\neg\varphi$) does not coincide with the negation of the semantics (there is no witness for φ); thus substituting a subformula and substituting its negation may return different results.

Semantics. Fix a state $q \in Q$, and a valuation $\chi\colon \mathcal{V} \cup \mathsf{Agt} \to \mathsf{Strat}(q)$. We inductively define the semantics of a subformula α of a formula of SL[BG]$^\flat$ at q under valuation χ, requiring $\mathsf{free}(\alpha) \subseteq \mathsf{dom}(\chi)$. We omit the easy cases of boolean combinations and atomic propositions.

Given a mapping $\sigma\colon \mathsf{Agt} \to \mathcal{V}$, the semantics of strategy assignments is defined as follows:

$$\mathcal{G}, q \models_\chi \mathsf{assign}(\sigma).\ \psi \quad \Leftrightarrow \quad \mathcal{G}, q \models_{\chi[A \in \mathsf{Agt} \mapsto \chi(\sigma(A))]} \psi.$$

Notice that, writing $\chi' = \chi[A \in \mathsf{Agt} \mapsto \chi(\sigma(A))]$, we have $\mathsf{free}(\psi) \subseteq \mathsf{dom}(\chi')$ if $\mathsf{free}(\alpha) \subseteq \mathsf{dom}(\chi)$, so that our inductive definition is sound.



$$\varphi = \forall y.\exists z.\forall x_A.\forall x_B.\ \bigvee \begin{cases} \mathsf{assign}(\square \mapsto x_A; \bigcirc \mapsto y; \diamond \mapsto z).\ \mathbf{F}\, p_1 \\ \mathsf{assign}(\square \mapsto x_B; \bigcirc \mapsto y; \diamond \mapsto z).\ \mathbf{F}\, p_2 \end{cases}$$

Figure 1 A game and a SL[BG] formula.

We now consider path formulas $\psi = \mathbf{X}\,\psi_1$ and $\psi = \psi_1\,\mathbf{U}\,\psi_2$. Since $\mathsf{Agt} \subseteq \mathsf{free}(\psi) \subseteq \mathsf{dom}(\chi)$, the valuation χ induces a unique outcome $\mathsf{out}(q,\chi) = (q_i)_{i\in\mathbb{N}}$ from q. For $n \in \mathbb{N}$, we write $\mathsf{out}_n(q,\chi) = (q_i)_{i\leq n}$, and define $\chi_{\overrightarrow{n}}$ as the valuation obtained by shifting all the strategies in the image of χ by $\mathsf{out}_n(q,\chi)$. Under the same conditions, we also define $q_{\overrightarrow{n}} = \mathsf{last}(\mathsf{out}_n(q,\chi))$. We then set

$$\begin{aligned} \mathcal{G}, q \models_\chi \mathbf{X}\,\psi_1 \quad &\Leftrightarrow \quad \mathcal{G}, q_{\overrightarrow{1}} \models_{\chi_{\overrightarrow{1}}} \psi_1 \\ \mathcal{G}, q \models_\chi \psi_1\,\mathbf{U}\,\psi_2 \quad &\Leftrightarrow \quad \exists k \in \mathbb{N}.\ \mathcal{G}, q_{\overrightarrow{k}} \models_{\chi_{\overrightarrow{k}}} \psi_2 \quad \text{and} \quad \forall 0 \leq j < k.\ \mathcal{G}, q_{\overrightarrow{j}} \models_{\chi_{\overrightarrow{j}}} \psi_1. \end{aligned}$$

In the sequel, we use classical shorthands, such as $\top$ for $p \vee \neg p$ (for any $p \in \mathsf{AP}$), $\mathbf{F}\,\psi$ for $\top\,\mathbf{U}\,\psi$ (*eventually* ψ), and $\mathbf{G}\,\psi$ for $\neg\mathbf{F}\,\neg\psi$ (*always* ψ). It remains to define the semantics of the strategy quantifiers. This is actually what this paper is all about. We provide here the original semantics, and discuss alternatives in the following sections:

$$\mathcal{G}, q \models_\chi \exists x.\varphi \quad \Leftrightarrow \quad \exists \delta \in \mathsf{Strat}(q).\ \mathcal{G}, q \models_{\chi[x\mapsto\delta]} \varphi.$$

▶ **Example 1.** *We consider the (turn-based) game $\mathcal{G}$ is depicted on Fig. 1. We name the players after the shape of the state they control. The SL[BG] formula φ to the right of Fig. 1 has four quantified variables and two goals. We show that this formula evaluates to true at q_0: fix a strategy δ_y (to be played by player ○); because $\mathcal{G}$ is turn-based, we identify the actions of the owner of a state with the resulting target state, so that $\delta_y(q_0q_1)$ will be either p_1 or p_2. We then define strategy δ_z (to be played by ◇) as $\delta_z(q_0q_2) = \delta_y(q_0q_1)$. Then clearly, for any strategy assigned to player □, one of the goals of formula φ holds true, so that φ itself evaluates to true.*

Subclasses of SL[BG]. Because of the high complexity and subtlety of reasoning with SL and SL[BG], several restrictions of SL[BG] have been considered in the literature [17, 20, 21], by adding further restrictions to boolean combinations in the grammar defining the syntax:

- SL[1G] restricts SL[BG] to a unique goal. SL[1G]$^\flat$ is then defined from the grammar of SL[BG]$^\flat$ by setting $\xi ::= \beta$ in the grammar;
- the larger fragment SL[CG] allows for *conjunctions* of goals. SL[CG]$^\flat$ corresponds to formulas defined with $\xi ::= \xi \wedge \xi \mid \beta$;
- similarly, SL[DG] only allows *disjunctions* of goals, i.e. $\xi ::= \xi \vee \xi \mid \beta$;
- finally, SL[AG] mixes conjunctions and disjunctions in a restricted way. Goals in SL[AG]$^\flat$ can be combined using the following grammar: $\xi ::= \beta \wedge \xi \mid \beta \vee \xi \mid \beta$.

In the sequel, we write a generic SL[BG]$^\flat$ formula φ as $(Q_i x_i)_{1\leq i\leq l}.\ \xi(\beta_j.\ \psi_j)_{j\leq n}$ where:

- $(Q_i x_i)_{i\leq l}$ is a block of quantifications, with $\{x_i \mid 1 \leq i \leq l\} \subseteq \mathcal{V}$ and $Q_i \in \{\exists, \forall\}$, for every $1 \leq i \leq l$;
- $\xi(g_1, ..., g_n)$ is a boolean combination of its arguments;
- for all $1 \leq j \leq n$, $\beta_j.\ \psi_j$ is a goal: β_j is a full assignment and ψ_j is an LTL formula.

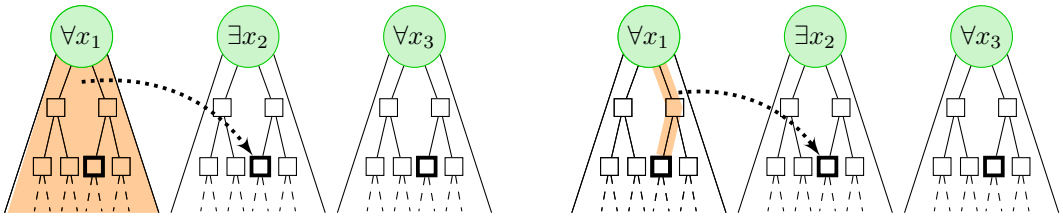


Figure 2 Classical (left) vs elementary (right) dependences for a formula $\forall x_1.\ \exists x_2.\ \forall x_3.\ \xi$.

3 Strategy dependences

We now follow the framework of [18, 21] and define the semantics of $\mathsf{SL[BG]}^\flat$ in terms of *dependence maps*. This approach provides a fine way of controlling how *existentially-quantified* strategies depend on other strategies (in a quantifier block). Using dependence maps, we can limit such dependences.

Dependence maps. Consider an $\mathsf{SL[BG]}^\flat$ formula $\varphi = (Q_i x_i)_{1\leq i\leq l}.\ \xi(\beta_j.\ \varphi_j)_{j\leq n}$, assuming w.l.o.g. that $\{x_i \mid 1 \leq i \leq l\} = \mathcal{V}$. We let $\mathcal{V}^\forall = \{x_i \mid Q_i = \forall\} \subseteq \mathcal{V}$ be the set of universally-quantified variables of φ. A function $\theta\colon \mathsf{Strat}^{\mathcal{V}^\forall} \to \mathsf{Strat}^{\mathcal{V}}$ is a *φ-map* (or *map* when φ is clear from the context) if $\theta(w)(x_i)(\rho) = w(x_i)(\rho)$ for any $w \in \mathsf{Strat}^{\mathcal{V}^\forall}$, any $x_i \in \mathcal{V}^\forall$, and any history ρ. In other words, $\theta(w)$ extends w to $\mathcal{V}$. This general notion allows any existentially-quantified variable to depend on *all* universally-quantified ones (dependence on existentially-quantified variables is implicit: all existentially-quantified variables are assigned through a single map, hence they all depend on the others); we add further restrictions later on. Using maps, we may then define new semantics for $\mathsf{SL[BG]}^\flat$: generally speaking, formula $\varphi = (Q_i x_i)_{1\leq i\leq l}.\ \xi(\beta_j.\ \varphi_j)_{j\leq n}$ holds true if there exists a φ-map θ such that, for any $w\colon \mathcal{V}^\forall \to \mathsf{Strat}$, the valuation $\theta(w)$ makes $\xi(\beta_j.\ \varphi_j)_{j\leq n}$ hold true.

Classic maps are dependence maps in which the order of quantification is respected:

$$\forall w_1, w_2 \in \mathsf{Strat}^{\mathcal{V}^\forall}.\ \forall x_i \in \mathcal{V}\setminus\mathcal{V}^\forall.\\ \big(\forall x_j \in \mathcal{V}^\forall \cap \{x_j \mid j < i\}.\ w_1(x_j) = w_2(x_j)\big) \Rightarrow \big(\theta(w_1)(x_i) = \theta(w_2)(x_i)\big). \quad \text{(C)}$$

In words, if w_1 and w_2 coincide on $\mathcal{V}^\forall \cap \{x_j \mid j < i\}$, then $\theta(w_1)$ and $\theta(w_2)$ coincide on x_i.

Elementary maps [18, 17] have to satisfy a more restrictive condition: for those maps, the value of an existentially-quantified strategy at any history ρ may only depend on the value of earlier universally-quantified strategies *along* ρ. This may be written as:

$$\forall w_1, w_2 \in \mathsf{Strat}^{\mathcal{V}^\forall}.\ \forall x_i \in \mathcal{V}\setminus\mathcal{V}^\forall.\ \forall \rho \in \mathsf{Hist}.\\ \big(\forall x_j \in \mathcal{V}^\forall \cap \{x_k \mid k < i\}.\ \forall \rho' \in \mathsf{Pref}(\rho)\cup\{\rho\}.\ w_1(x_j)(\rho') = w_2(x_j)(\rho')\big) \Rightarrow\\ \big(\theta(w_1)(x_i)(\rho) = \theta(w_2)(x_i)(\rho)\big). \quad \text{(E)}$$

In this case, for any history ρ, if two valuations w_1 and w_2 of the universally-quantified variables coincide on the variables quantified before x_i all along ρ, then $\theta(w_1)(x_i)$ and $\theta(w_2)(x_i)$ have to coincide at ρ.

The difference between both kinds of dependences is illustrated on Fig. 2: for classic maps, the existentially-quantified strategy x_2 may depend on the whole strategy x_1, while it may only depend on the value of x_1 along the current history for elementary maps. Notice that a map satisfying (E) also satisfies (C).

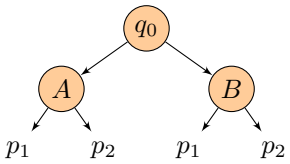


Figure 3 A game $\mathcal{G}$ and an $\mathsf{SL[BG]}^\flat$ formula φ such that $\mathcal{G}, q_0 \not\models^E \varphi$ and $\mathcal{G}, q_0 \not\models^E \neg\varphi$.

Satisfaction relations. Pick a formula $\varphi = (Q_i x_i)_{1\leq i\leq l}.\ \xi\big(\beta_j.\ \varphi_j\big)_{j\leq n}$ in $\mathsf{SL[BG]}^\flat$. We define:

$$\mathcal{G}, q \models^C \varphi \quad \text{iff} \quad \exists \theta \text{ satisfying (C)}.\ \forall w \in \mathsf{Strat}^{\mathcal{V}^\forall}.\ \mathcal{G}, q \models_{\theta(w)} \xi\big(\beta_j \varphi_j\big)_{j\leq n}$$

As explained above, this actually corresponds to the usual semantics of $\mathsf{SL[BG]}^\flat$ as given in Section 2 [18, Theorem 4.6]. When $\mathcal{G}, q \models^C \varphi$, a map θ satisfying the conditions above is called a *C-witness* of φ for $\mathcal{G}$ and q. Similarly, we define the *elementary semantics* [18] as:

$$\mathcal{G}, q \models^E \varphi \quad \text{iff} \quad \exists \theta \text{ satisfying (E)}.\ \forall w \in \mathsf{Strat}^{\mathcal{V}^\forall}.\ \mathcal{G}, q \models_{\theta(w)} \xi\big(\beta_j \varphi_j\big)_{j\leq n}$$

Again, when such a map exists, it is called an E-witness. Notice that since Property (E) implies Property (C), we have $\mathcal{G}, q \models^E \varphi \Rightarrow \mathcal{G}, q \models^C \varphi$ for any $\varphi \in \mathsf{SL[BG]}^\flat$. This corresponds to the intuition that it is harder to satisfy a $\mathsf{SL[BG]}^\flat$ formula when dependences are more restricted. The contrapositive statement then raises questions about the negation of formulas.

The syntactic vs. semantic negations. If $\varphi = (Q_i x_i)_{1\leq i\leq l} \xi(\beta_j \varphi_j)_{j\leq n}$ is an $\mathsf{SL[BG]}^\flat$ formula, its syntactic negation $\neg\varphi$ is the formula $(\overline{Q}_i x_i)_{i\leq l}(\neg\xi)(\beta_j\varphi_j)_{j\leq n}$, where $\overline{Q}_i = \exists$ if $Q_i = \forall$ and $\overline{Q}_i = \forall$ if $Q_i = \exists$. Looking at the definitions of $\models^C$ and $\models^E$, it could be the case that e.g. $\mathcal{G}, q \models^C \varphi$ and $\mathcal{G}, q \models^C \neg\varphi$: this only requires the existence of two adequate maps. However, since $\models^C$ and $\models$ coincide, and since $\mathcal{G}, q \models \varphi \Leftrightarrow \mathcal{G}, q \not\models \neg\varphi$ in the usual semantics, we get $\mathcal{G}, q \models^C \varphi \Leftrightarrow \mathcal{G}, q \not\models^C \neg\varphi$. Also, since $\mathcal{G}, q \models^E \varphi \Rightarrow \mathcal{G}, q \models^C \varphi$, we also get $\mathcal{G}, q \models^E \varphi \Rightarrow \mathcal{G}, q \not\models^E \neg\varphi$. As we now show, the converse implication holds for $\mathsf{SL[1G]}^\flat$, but may fail to hold for $\mathsf{SL[BG]}^\flat$.

▶ **Proposition 1.** *There exist a (one-player) game $\mathcal{G}$ with initial state q_0 and a formula $\varphi \in$ SL[BG]$^\flat$ such that $\mathcal{G}, q_0 \not\models^E \varphi$ and $\mathcal{G}, q_0 \not\models^E \neg\varphi$.*

Proof. Consider the formula and the one-player game of Fig. 3. We start by proving that $\mathcal{G}, q_0 \not\models^E \varphi$. For a contradiction, assume that a witness map θ satisfying (E) exists, and pick any valuation w for the universal variable x. First, for the first goal in the conjunction to be fulfilled, the strategy assigned to y must play to B from q_0. We abbreviate this as $\theta(w)(y)(q_0) = B$ in the sequel. Now, consider two valuations w_1 and w_2 such that $w_1(x)(q_0) = w_2(x)(q_0) = A$ and $w_1(x)(q_0 \cdot B) = w_2(x)(q_0 \cdot B)$, but such that $w_1(x)(q_0 \cdot A) = p_1$ and $w_2(x)(q_0 \cdot A) = p_2$. In order to fulfill the second goal under both valuations w_1 and w_2, we must have $\theta(w_1)(y)(q_0 \cdot B) = p_1$ and $\theta(w_2)(y)(q_0 \cdot B) = p_2$. But this violates Property (E): since $w_1(x)$ and $w_2(x)$ coincide on q_0 and on $q_0 \cdot B$, we must have $\theta(w_1)(y)(q_0 \cdot B) = \theta(w_2)(y)(q_0 \cdot B)$.

We now prove that $\mathcal{G}, q_0 \not\models^E \neg\varphi$. Indeed, following the previous discussion, we easily get that $\mathcal{G}, q_0 \models^C \varphi$, by letting $\theta(w)(y)(q_0) = B$ and $\theta(w)(y)(q_0 \cdot B) = w(x)(q_0 \cdot A)$ if $w(x)(q_0) = A$, and $\theta(w)(y)(q_0 \cdot B) = w(x)(q_0 \cdot B)$ if $w(x)(q_0) = B$. As explained above, this entails $\mathcal{G}, q_0 \not\models^C \neg\varphi$, and $\mathcal{G}, q_0 \not\models^E \neg\varphi$. ◀

▶ **Proposition 2.** *For any game $\mathcal{G}$ with initial state q_0, and any formula $\varphi \in$ SL[1G]$^\flat$, it holds $\mathcal{G}, q_0 \models^E \varphi \Leftrightarrow \mathcal{G}, q_0 \not\models^E \neg\varphi$.*

Sketch of proof. This result follows from [18, Corollary 4.21], which states that $\models^C$ and $\models^E$ coincide on SL[1G]. Because it is central in our approach, we sketch a direct proof here using similar ingredients: it consists in encoding the problem whether $\mathcal{G}, q_0 \models^E \varphi$ into a two-player turn-based game with a parity-winning objective.

The construction is as follows: the interaction between existential and universal quantifications of the formula is integrated into the game structure, replacing each state of $\mathcal{G}$ with a tree-shaped subgame where Player $P_\exists$ selects existentially-quantified actions and Player $P_\forall$ selects universally-quantified ones. The unique goal of the formula is then incorporated into the game via a deterministic parity automaton, yielding a two-player turn-based parity game. We then show that $\mathcal{G}, q_0 \models^E \varphi$ if, and only if, Player $P_\exists$ has a winning strategy in the resulting turn-based parity game, while $\mathcal{G}, q_0 \models^E \neg\varphi$ if, and only if, Player $P_\forall$ has a winning strategy. Those equivalences hold for the elementary semantics because memoryless strategies are sufficient in parity games. Proposition 2 then follows by determinacy of those games [11, 23]. ◀

Note that the construction of the parity game gives an effective algorithm for the model-checking problem of SL[1G]$^\flat$, which runs in time doubly-exponential in the size of the formula, and polynomial in the size of the game structure; we recover the result of [18] for that problem.

Comparison of $\models^C$ and $\models^E$. A consequence of Prop. 2 is that $\models^C$ and $\models^E$ coincide on SL[1G]$^\flat$ (Corollary 4.21 of [18]). However, when considering larger fragments, the satisfaction relations are distinct (see the proof of Prop. 1 for a candidate formula in SL[CG]$^\flat$):

▶ **Proposition 3.** *The relations $\models^C$ and $\models^E$ differ on SL[CG]$^\flat$, as well as on SL[DG]$^\flat$.*

▶ Remark. Proposition 3 contradicts the claim in [20] that $\models^E$ and $\models^C$ coincide on SL[CG] (and SL[DG]). Indeed, in [20], the satisfaction relation for SL[DG] and SL[CG] is encoded into a two-player game in pretty much the same way as we did in the proof of Prop. 2. While this indeed rules out dependences outside the current history, it also gives information to Player $P_\exists$ about the values (over prefixes of the current history) of strategies that are universally-quantified later in the quantification block. This proof technique works with SL[1G]$^\flat$ because the single goal can be encoded as a parity objective, for which memoryless strategies exist, so that the extra information is not crucial. In the next section, we investigate the role of this extra information for larger fragments of SL[BG]$^\flat$.

4 Timeline dependences

Following the discussion above, we introduce a new type of dependences between strategies (which we call *timeline dependences*). They allow strategies to also observe (and depend on) *all* other universally-quantified strategies on the strict prefix of the current history. For instance, for a block of quantifiers $\forall x_1.\, \exists x_2.\, \forall x_3$, the value of x_2 after history ρ may depend on the value of x_1 on ρ and its prefixes (as for elementary maps), but also on the value of x_3 on the (strict) prefixes of ρ. Such dependences are depicted on Fig. 4. We believe that such dependences are relevant in many situations, especially for reactive synthesis, since in this framework strategies really base their decisions on what they could observe along the current history.

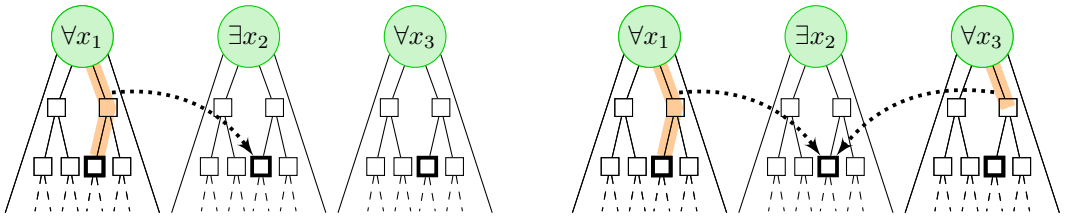


Figure 4 Elementary (left) vs timeline (right) dependences for a formula $\forall x_1.\ \exists x_2.\ \forall x_3.\ \xi$.

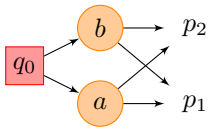


Figure 5 $\models^E$ and $\models^T$ differ on $\mathsf{SL[CG]}^\flat$.

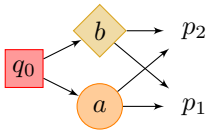


Figure 6 $\models^E$ and $\models^T$ differ on $\mathsf{SL[DG]}^\flat$.

Formally, a map θ is a *timeline map* if it satisfies the following condition:

$$\forall w_1, w_2 \in \mathsf{Strat}^{\mathcal{V}^\forall}.\ \forall x_i \in \mathcal{V} \setminus \mathcal{V}^\forall.\ \forall \rho \in \mathsf{Hist}.$$
$$\left(\begin{array}{r} \forall x_j \in \mathcal{V}^\forall \cap \{x_k \mid k < i\}.\ \forall \rho' \in \mathsf{Pref}(\rho) \cup \{\rho\}.\ w_1(x_j)(\rho) = w_2(x_j)(\rho) \\ \wedge\ \forall x_j \in \mathcal{V}^\forall.\ \forall \rho' \in \mathsf{Pref}(\rho).\ w_1(x_j)(\rho) = w_2(x_j)(\rho) \end{array} \right) \Rightarrow$$
$$\big(\theta(w_1)(x_i)(\rho) = \theta(w_2)(x_i)(\rho)\big). \quad \text{(T)}$$

Using those maps, we introduce the *timeline semantics* of $\mathsf{SL[BG]}^\flat$:

$$\mathcal{G}, q \models^T \varphi \quad \text{iff} \quad \exists \theta \text{ satisfying (T)}.\ \forall w \in \mathsf{Strat}^{\mathcal{V}^\forall}.\ \mathcal{G}, q \models_{\theta(w)} \xi\big(\beta_j \varphi_j\big)_{j \le n}$$

Such a map, if any, is called a T-witness of φ for $\mathcal{G}$ and q. As in the previous section, it is easily seen that Property (E) implies Property (T), so that an E-witness is also a T-witness, and $\mathcal{G}, q \models^E \varphi \Rightarrow \mathcal{G}, q \models^T \varphi$ for any formula $\varphi \in \mathsf{SL[BG]}^\flat$.

▶ **Example 2.** *Consider again the game of Fig 1 in Section 2. We have seen that* $\mathcal{G}, q_0 \models^C \varphi$ *in Section 2, and that* $\mathcal{G}, q_0 \not\models^E \varphi$ *in the proof of Prop. 3. With timeline dependences, we have* $\mathcal{G}, q_0 \models^T \varphi$. *Indeed, now* $\theta(w)(z)(q_0 \cdot q_2)$ *may depend on* $w(x_A)(q_0)$ *and* $w(x_B)(q_0)$*: we could then have e.g.* $\theta(w)(z)(q_0 \cdot q_2) = p_1$ *when* $w(x_A)(q_0) = q_2$*, and* $\theta(w)(z)(q_0 \cdot q_2) = p_2$ *when* $w(x_A)(q_0) = q_1$. *It is easily checked that this map is a T-witness of* φ *for* q_0.

Comparison of $\models^E$ and $\models^T$. As explained at the end of Section 3, the proof of Prop. 2 actually shows the following result:

▶ **Proposition 4.** *For any game* $\mathcal{G}$ *with initial state* q_0*, and any formula* $\varphi \in \mathsf{SL[1G]}^\flat$*, it holds* $\mathcal{G}, q_0 \models^E \varphi \Leftrightarrow \mathcal{G}, q_0 \models^T \varphi$.

We now prove that this does not extend to $\mathsf{SL[CG]}^\flat$ and $\mathsf{SL[DG]}^\flat$:

▶ **Proposition 5.** *The relations* $\models^E$ *and* $\models^T$ *differ on* $\mathsf{SL[CG]}^\flat$*, as well as on* $\mathsf{SL[DG]}^\flat$.

Proof. For $\mathsf{SL[CG]}^\flat$, we consider the game structure of Fig. 5, and formula

$$\varphi_C = \exists y.\ \forall x_A.\ \exists x_B.\ \bigwedge \begin{cases} \mathsf{assign}(\bigcirc \mapsto y;\ \square \mapsto x_A).\ \mathbf{F}\, p_1 \\ \mathsf{assign}(\bigcirc \mapsto y;\ \square \mapsto x_B).\ \mathbf{F}\, p_2 \end{cases}$$

We first notice that $\mathcal{G}, q_0 \not\models^E \varphi_C$: indeed, in order to satisfy the first goal under any choice of x_A, the strategy for y has to point to p_1 from both a and b. But then no choice of x_B will make the second goal true.

On the other hand, considering the timeline semantics, strategy y after $q_0 \cdot a$ and $q_0 \cdot b$ may depend on the choice of x_A in q_0. When $w(x_A)(q_0) = a$, we let $\theta(w)(y)(q_0 \cdot a) = p_1$ and $\theta(w)(y)(q_0 \cdot b) = p_2$ and $\theta(w)(x_B)(q_0) = b$, which makes both goals hold true. Conversely, if $w(x_A)(q_0) = b$, then we let $\theta(w)(y)(q_0 \cdot b) = p_1$ and $\theta(w)(y)(q_0 \cdot a) = p_2$ and $\theta(w)(x_B)(q_0) = a$.

For $\mathsf{SL[DG]}^\flat$, we consider the game of Fig. 6, and easily prove that formula φ_D below has a T-witness but no E-witness:

$$\varphi_D = \exists y.\ \forall x_A.\ \forall x_B.\ \forall z.\ \bigvee \begin{cases} \mathsf{assign}(\bigcirc \mapsto y; \square \mapsto x_A; \diamond \mapsto z).\ \mathbf{F}\, p_1 \\ \mathsf{assign}(\bigcirc \mapsto y; \square \mapsto x_B; \diamond \mapsto z).\ \mathbf{F}\, p_2 \end{cases}$$ ◀

The syntactic vs. semantic negations. While both semantics differ, we now prove that the situation w.r.t. the syntactic vs. semantic negations is similar. First, following Prop. 4 and 2, the two negations coincide on $\mathsf{SL[1G]}^\flat$ under the timeline semantics. Moreover:

▶ **Proposition 6.** *For any formula φ in $\mathsf{SL[BG]}^\flat$, for any game $\mathcal{G}$ and any state q_0, we have $\mathcal{G}, q_0 \models^T \varphi \Rightarrow \mathcal{G}, q_0 \not\models^T \neg\varphi$.*

Sketch of proof. Write $\varphi = (Q_i x_i)_{1 \le i \le l} \xi(\beta_j \varphi_j)_{j \le n}$. For a contradiction, assume that there exist two maps θ and $\overline{\theta}$ witnessing $\mathcal{G}, q_0 \models^T \varphi$ and $\mathcal{G}, q_0 \models^T \neg\varphi$, respectively. Then for any strategy valuations w and $\overline{w}$ for $\mathcal{V}^\forall$ and $\mathcal{V}^\exists$, we have that $\mathcal{G}, q_0 \models_{\theta(w)} \xi(\beta_j \varphi_j)_j$ and $\mathcal{G}, q_0 \models_{\overline{\theta}(\overline{w})} \neg\xi(\beta_j \varphi_j)_j$. We can then inductively (on histories and on the sequence of quantified variables) build a strategy valuation χ on $\mathcal{V}$ such that $\theta(\chi_{|\mathcal{V}^\forall}) = \overline{\theta}(\chi_{|\mathcal{V}^\exists}) = \chi$. Then under valuation χ, both $\xi(\beta_j \varphi_j)_j$ and $\neg\xi(\beta_j \varphi_j)_j$ hold in q_0, which is impossible. ◀

▶ **Proposition 7.** *There exists a formula $\varphi \in \mathsf{SL[BG]}^\flat$, a (turn-based) game $\mathcal{G}$ and a state q_0 such that $\mathcal{G}, q_0 \not\models^T \varphi$ and $\mathcal{G}, q_0 \not\models^T \neg\varphi$.*

5 The fragment $\mathsf{SL[EG]}^\flat$

In this section, we focus on the timeline semantics $\models^T$. We exhibit a fragment[1] $\mathsf{SL[EG]}^\flat$ of $\mathsf{SL[BG]}^\flat$, containing $\mathsf{SL[CG]}^\flat$ and $\mathsf{SL[DG]}^\flat$, for which the syntactic and semantic negations coincide, and for which we prove model-checking is in *2*-EXPTIME:

▶ **Theorem 8.** *For any $\varphi \in \mathsf{SL[EG]}^\flat$ and any state q_0, it holds: $\mathcal{G}, q_0 \models^T \varphi \Leftrightarrow \mathcal{G}, q_0 \not\models^T \neg\varphi$. Moreover, model checking $\mathsf{SL[EG]}^\flat$ for the timeline semantics is 2-EXPTIME-complete.*

5.1 Semi-stable sets.

For $n \in \mathbb{N}$, we let $\{0,1\}^n$ be the set of mappings from $[1,n]$ to $\{0,1\}$. We write $\mathbf{0}^n$ (or $\mathbf{0}$ if the size n is clear) for the function that maps all integers in $[1,n]$ to 0, and $\mathbf{1}^n$ (or $\mathbf{1}$) for the function that maps $[1,n]$ to 1. For $f, g \in \{0,1\}^n$, we define:

$$\overline{f}\colon i \mapsto 1 - f(i) \qquad f \curlywedge g\colon i \mapsto \min\{f(i), g(i)\} \qquad f \curlyvee g\colon i \mapsto \max\{f(i), g(i)\}.$$

[1] We name our fragment $\mathsf{SL[EG]}^\flat$ as it comes as a natural continuation after fragments $\mathsf{SL[AG]}^\flat$ [21], $\mathsf{SL[BG]}^\flat$ [18], and $\mathsf{SL[CG]}^\flat$ and $\mathsf{SL[DG]}^\flat$ [20].

We then introduce the notion of semi-stable sets, on which the definition of $\mathsf{SL[EG]}^\flat$ relies: a set $F^n \subseteq \{0,1\}^n$ is *semi-stable* if for any f and g in F^n, it holds that

$$\forall s \in \{0,1\}^n. \qquad (f \curlywedge s) \curlyvee (g \curlywedge \overline{s}) \in F^n \text{ or } (g \curlywedge s) \curlyvee (f \curlywedge \overline{s}) \in F^n.$$

▶ **Example 3.** *Obviously, the set $\{0,1\}^n$ is semi-stable, as well as the empty set. For $n = 2$, the set $\{(0,1),(1,0)\}$ is easily seen not to be semi-stable: taking $f = (0,1)$ and $g = (1,0)$ with $s = (1,0)$, we get $(f \curlywedge s) \curlyvee (g \curlywedge \overline{s}) = (0,0)$ and $(g \curlywedge s) \curlyvee (f \curlywedge \overline{s}) = (1,1)$. Similarly, $\{(0,0),(1,1)\}$ is not semi-stable. Any other subset of $\{0,1\}^2$ is semi-stable.*

We then define

$$\begin{aligned} \mathsf{SL[EG]}^\flat \ni \varphi &::= \forall x.\varphi \mid \exists x.\varphi \mid \xi & \xi &::= F^n((\beta_i)_{1\le i\le n}) \\ \beta &::= \mathsf{assign}(\sigma).\ \psi & \psi &::= \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\,\psi \mid \psi\,\mathbf{U}\,\psi \mid p \end{aligned}$$

where F^n ranges over semi-stable subsets of $\{0,1\}^n$, for all $n \in \mathbb{N}$. The semantics of the operator F^n is defined as

$$\mathcal{G}, q \models_\chi F^n((\beta_i)_{i\le n}) \quad \Leftrightarrow \quad \exists f \in F^n.\ \forall 1 \le i \le n.\ (f(i) = 1 \text{ iff } \mathcal{G}, q \models_\chi \beta_i).$$

Notice that if F^n would range over all subsets of $\{0,1\}^n$, then this definition would exactly correspond to $\mathsf{SL[BG]}^\flat$. Similarly, the case where $F^n = \{\mathbf{1}^n\}$ corresponds to $\mathsf{SL[CG]}^\flat$, while $F^n = \{0,1\}^n \setminus \{\mathbf{0}^n\}$ gives rise to $\mathsf{SL[DG]}^\flat$.

▶ **Example 4.** *Consider the following formula, expressing the existence of a Nash equilibrium for two players with respective LTL objectives ψ_1 and ψ_2:*

$$\exists x_1.\exists x_2.\forall y_1.\forall y_2.\ \bigwedge \begin{cases} (\mathsf{assign}(A_1 \mapsto y_1; A_2 \mapsto x_2).\psi_1) \Rightarrow (\mathsf{assign}(A_1 \mapsto x_1; A_2 \mapsto x_2).\psi_1) \\ (\mathsf{assign}(A_1 \mapsto x_1; A_2 \mapsto y_2).\psi_2) \Rightarrow (\mathsf{assign}(A_1 \mapsto x_1; A_2 \mapsto x_2).\psi_2) \end{cases}$$

This formula has four goals, and it corresponds to the set

$$F^4 = \{(a,b,c,d) \in \{0,1\}^4 \mid a \le b \text{ and } c \le d\}$$

Taking $f = (1,1,0,0)$ and $g = (0,0,1,1)$, with $s = (1,0,1,0)$ we have $(f \curlywedge s) \curlyvee (g \curlywedge \overline{s}) = (1,0,0,1)$ and $(g \curlywedge s) \curlyvee (f \curlywedge \overline{s}) = (0,1,1,0)$, none of which is in F^4. Hence our formula is not (syntactically) in SL[EG]$^\flat$.

The definition of SL[EG] may look artificial. The main reason why we work with SL[EG] is that it is maximal for the first claim of Theorem 8 (see Prop. 11). But as the next result shows, it is actually a large fragment encompassing SL[AG] (hence also SL[CG] and SL[DG]):

▶ **Proposition 9.** *SL[EG]$^\flat$ contains SL[AG]$^\flat$. The inclusion is strict (syntactically).*

5.2 Defining quasi-orders from semi-stable sets.

For $F^n \subseteq \{0,1\}^n$, we write $\overline{F^n}$ for the complement of F^n. Fix such a set F^n, and pick $s \in \{0,1\}^n$. For any $h \in \{0,1\}^n$, we define

$$\mathbb{F}^n(h,s) = \{h' \in \{0,1\}^n \mid (h \curlywedge s) \curlyvee (h' \curlywedge \overline{s}) \in F^n\}$$
$$\overline{\mathbb{F}^n}(h,s) = \{h' \in \{0,1\}^n \mid (h \curlywedge s) \curlyvee (h' \curlywedge \overline{s}) \in \overline{F^n}\}$$

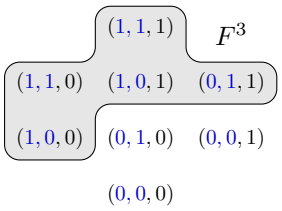
Trivially $\mathbb{F}^n(h,s) \cap \overline{\mathbb{F}^n}(h,s) = \emptyset$ and $\mathbb{F}^n(h,s) \cup \overline{\mathbb{F}^n}(h,s) = \{0,1\}^n$. If we assume F^n to be semi-stable, then the family $(\mathbb{F}^n(h,s))_{h\in\{0,1\}^n}$ enjoys the following property:

▶ **Lemma 10.** *Fix a semi-stable set F^n and $s \in \{0,1\}^n$. For any $h_1, h_2 \in \{0,1\}^n$, either $\mathbb{F}^n(h_1, s) \subseteq \mathbb{F}^n(h_2, s)$ or $\mathbb{F}^n(h_2, s) \subseteq \mathbb{F}^n(h_1, s)$.*

Given a semi-stable set F^n and $s \in \{0,1\}^n$, we can use the inclusion relation of Lemma 10 to define a relation $\preceq_s^{F^n}$ (written $\preceq_s$ when F^n is clear) over the elements of $\{0,1\}^n$. It is defined as follows: $h_1 \preceq_s h_2$ if, and only if, $\mathbb{F}^n(h_1, s) \subseteq \mathbb{F}^n(h_2, s)$.

This relation is a quasi-order: its reflexiveness and transitivity both follow from the reflexiveness and transitivity of the inclusion relation $\subseteq$. By Lemma 10, this quasi-order is total. Intuitively, $\preceq_s$ orders the elements of $\{0,1\}^n$ based on how "easy" it is to complete their restriction to s so that the completion belongs to F^n. In particular, only the indices on which s take value 1 are used to check whether $h_1 \preceq_s h_2$: given $h_1, h_2 \in \{0,1\}^n$ such that $(h_1 \curlywedge s) = (h_2 \curlywedge s)$, we have $\mathbb{F}(h_1, s) = \mathbb{F}(h_2, s)$, and $h_1 \equiv_s h_2$.

▶ **Example 5.** *Consider the set $F^3 = \{(1,0,0), (1,1,0), (1,0,1), (0,1,1), (1,1,1)\}$ represented on the figure below, and which can be shown to be semi-stable. Fix $s = (1,1,0)$. Then $\mathbb{F}^3((0,1,\star), s) = \{0,1\}^2 \times \{1\}$, while $\mathbb{F}^3((1,1,\star), s) = \mathbb{F}^3((1,0,\star), s) = \{0,1\}^3$ and $\mathbb{F}^3((0,0,\star), s) = \emptyset$. It follows that $(0,0,\star) \preceq_s (0,1,\star) \preceq_s (1,0,\star) \equiv_s (1,1,\star)$.*



5.3 Sketch of proof of Theorem 8

The approach we used in Prop 2 does not extend in general to formulas with several goals. Consider for instance formula $(Q_i x_i)_{i \leq l}(\beta_1.\psi_1 \Leftrightarrow \beta_2.\psi_2)$: if at some points the two goals give rise to two different outcomes (hence to two different subgames), the winning objectives in one subgame depends on what is achieved in the other subgame.

SL[EG]$^\flat$ has been designed to prevent such situations: when two (or more) outcomes are available at a given position, each *subgame* can be assigned an independent winning objective. This objective can be obtained from the quasi-orders $\preceq_s$ associated with the SL[EG]$^\flat$ formula being considered. Consider again Example 5: associating the set F^3 with three goals β_1, β_2 and β_3, we get a formula in SL[EG]$^\flat$. Assume that the moves selected by the players give rise to the same transition for β_1 and β_2, and to a different transition for β_3; then in the subgame reached when following the transition of β_1 and β_2 (hence with $s = (1,1,0)$), the optimal way of fulfilling goals β_1 and β_2 is given by $(0,0,\star) \preceq_s (0,1,\star) \preceq_s (1,0,\star) \equiv_s (1,1,\star)$, independently of what may happen in the subgame reached by following the transition given by β_3.

We exploit this idea in our proof: first, in order to keep track of the truth values of the LTL formulas ψ_i of each goal, we define a family of parity automata, one for each subset of goals of the formula under scrutiny. A subgame, as considered above, is characterized by a state q of the original concurrent game, a state d_p of each of the parity automata, and a vector $s \in \{0,1\}^n$ defining which goals are still active. For each subgame, we can compute, by induction on s, the optimal set of goals that can be fulfilled from that configuration. The optimal strategies of both players in each subgame can be used to define (partial) optimal timeline dependence maps. We can then combine these partial maps together to get optimal dependence maps θ and $\overline{\theta}$; using similar arguments as for the proof of Prop. 6, we get a

valuation χ such that $\theta(\chi_{|\mathcal{V}^\forall}) = \chi = \overline{\theta}(\chi_{|\mathcal{V}^\exists})$, from which we deduce that exactly one of φ and $\neg\varphi$ holds.

5.4 Maximality of SL[EG]$^\flat$

Finally, we prove that SL[EG]$^\flat$ is, in a sense, maximal for the first property of Theorem 8:

▶ **Proposition 11.** *For any non-semi-stable boolean set $F^n \subseteq \{0,1\}^n$, there exists a SL[BG]$^\flat$ formula φ built on F^n, a game $\mathcal{G}$ and a state q_0 such that $\mathcal{G}, q_0 \not\models^T \neg\varphi$ and $\mathcal{G}, q_0 \not\models^T \varphi$.*

Whether SL[EG]$^\flat$ is also maximal for having a *2*-EXPTIME model-checking algorithm remains open. Actually, we do not know if SL[BG]$^\flat$ model checking is decidable under the timeline semantics. These questions will be part of our future works on this topic.

References

1. Thomas Ågotnes, Valentin Goranko, and Wojciech Jamroga. Alternating-time temporal logics with irrevocable strategies. In Dov Samet, editor, *Proceedings of the 11th Conference on Theoretical Aspects of Rationality and Knowledge (TARK'07)*, pages 15–24, 2007.
2. Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002. doi:10.1145/585265.585270.
3. Patricia Bouyer, Patrick Gardy, and Nicolas Markey. Weighted strategy logic with boolean goals over one-counter games. In Prahladh Harsha and G. Ramalingam, editors, *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, volume 45 of *LIPIcs*, pages 69–83. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.69.
4. Patricia Bouyer, Patrick Gardy, and Nicolas Markey. On the semantics of strategy logic. *Inf. Process. Lett.*, 116(2):75–79, 2016. doi:10.1016/j.ipl.2015.10.004.
5. Romain Brenguier, Jean-François Raskin, and Ocan Sankur. Assume-admissible synthesis. *Acta Inf.*, 54(1):41–83, 2017. doi:10.1007/s00236-016-0273-2.
6. Thomas Brihaye, Arnaud Da Costa Lopes, François Laroussinie, and Nicolas Markey. ATL with strategy contexts and bounded memory. In Sergei N. Artëmov and Anil Nerode, editors, *Logical Foundations of Computer Science, International Symposium, LFCS 2009, Deerfield Beach, FL, USA, January 3-6, 2009. Proceedings*, volume 5407 of *Lecture Notes in Computer Science*, pages 92–106. Springer, 2009. doi:10.1007/978-3-540-92687-0_7.
7. Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy logic. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings*, volume 4703 of *Lecture Notes in Computer Science*, pages 59–73. Springer, 2007. doi:10.1007/978-3-540-74407-8_5.
8. Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981. doi:10.1007/BFb0025774.
9. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, 2000.
10. Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The complexity of rational synthesis. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of

LIPIcs, pages 121:1–121:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.121.

11 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185392.

12 Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010. doi:10.1007/978-3-642-12002-2_16.

13 Valentin Goranko and Govert van Drimmelen. Complete axiomatization and decidability of alternating-time temporal logic. *Theoretical Computer Science*, 353(1-3):93–117, mar 2006.

14 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016. doi:10.1007/s10472-016-9508-8.

15 François Laroussinie and Nicolas Markey. Augmenting ATL with strategy contexts. *Inf. Comput.*, 245:98–123, 2015. doi:10.1016/j.ic.2014.12.020.

16 François Laroussinie, Nicolas Markey, and Ghassan Oreiby. On the expressiveness and complexity of ATL. *Logical Methods in Computer Science*, 4(2), 2008. doi:10.2168/LMCS-4(2:7)2008.

17 Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. What makes ATL* decidable? A decidable fragment of strategy logic. In Maciej Koutny and Irek Ulidowski, editors, *Proceedings of the 23rd International Conference on Concurrency Theory (CONCUR'12)*, volume 7454 of *Lecture Notes in Computer Science*, pages 193–208. Springer-Verlag, sep 2012.

18 Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. Reasoning about strategies: On the model-checking problem. *ACM Trans. Comput. Log.*, 15(4):34:1–34:47, 2014. doi:10.1145/2631917.

19 Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. Reasoning about strategies: on the satisfiability problem. *Logical Methods in Computer Science*, 13(1), 2017. doi:10.23638/LMCS-13(1:9)2017.

20 Fabio Mogavero, Aniello Murano, and Luigi Sauro. On the boundary of behavioral strategies. In *Proceedings of the 28th Annual Symposium on Logic in Computer Science (LICS'13)*, pages 263–272. IEEE Comp. Soc. Press, jun 2013.

21 Fabio Mogavero, Aniello Murano, and Luigi Sauro. A behavioral hierarchy of strategy logic. In Nils Bulling, Leendert W. N. van der Torre, Serena Villata, Wojtek Jamroga, and Wamberto Weber Vasconcelos, editors, *Computational Logic in Multi-Agent Systems - 15th International Workshop, CLIMA XV, Prague, Czech Republic, August 18-19, 2014. Proceedings*, volume 8624 of *Lecture Notes in Computer Science*, pages 148–165. Springer, 2014. doi:10.1007/978-3-319-09764-0_10.

22 Fabio Mogavero, Aniello Murano, and Moshe Y. Vardi. Reasoning about strategies. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 133–144. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.133.

23 Andrzej Mostowski. Games with forbidden positions. Research Report 78, University of Danzig, 1991.

24 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

25 Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *International Symposium on Programming, 5th Colloquium, Torino, Italy, April 6-8, 1982, Proceedings*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982. doi:10.1007/3-540-11494-7_22.

26 Farn Wang, Chung-Hao Huang, and Fang Yu. A temporal logic for the interaction of strategies. In Joost-Pieter Katoen and Barbara König, editors, *Proceedings of the 22nd International Conference on Concurrency Theory (CONCUR'11)*, volume 6901 of *Lecture Notes in Computer Science*, pages 466–481. Springer-Verlag, sep 2011.

Colouring Square-Free Graphs without Long Induced Paths

Serge Gaspers

UNSW Sydney and Data61, CSIRO, Australia
sergeg@cse.unsw.edu.au

Shenwei Huang

UNSW Sydney, Australia
dynamichuang@gmail.com

Daniël Paulusma

Durham University, Durham, UK
daniel.paulusma@durham.ac.uk

Abstract

The Colouring problem is to decide if the vertices of a graph can be coloured with at most k colours for a given integer k such that no two adjacent vertices are coloured alike. The complexity of Colouring is fully understood for graph classes characterized by one forbidden induced subgraph H. Despite a huge body of existing work, there are still major complexity gaps if two induced subgraphs H_1 and H_2 are forbidden. We let H_1 be the s-vertex cycle C_s and H_2 be the t-vertex path P_t. We show that Colouring is polynomial-time solvable for $s = 4$ and $t \leq 6$, which unifies several known results for Colouring on (H_1, H_2)-free graphs. Our algorithm is based on a novel decomposition theorem for (C_4, P_6)-free graphs without clique cutsets into homogeneous pairs of sets and a new framework for bounding the clique-width of a graph by the clique-width of its subgraphs induced by homogeneous pairs of sets. To apply this framework, we also need to use divide-and-conquer to bound the clique-width of subgraphs induced by homogeneous pairs of sets. To complement our positive result we also prove that Colouring is NP-complete for $s = 4$ and $t \geq 9$, which is the first hardness result on Colouring for (C_4, P_t)-free graphs.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Graph Colouring, Hereditary Graph Class, Clique-width, Cycle, Path

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.35

Funding Serge Gaspers is the recipient of an Australian Research Council (ARC) Future Fellowship (FT140100048) and acknowledges support under the ARC's Discovery Projects funding scheme (DP150101134). Daniël Paulusma is supported by Leverhulme Trust Grant RPG-2016-258.

Acknowledgements Initially we proved NP-hardness of Colouring for (C_4, P_{16})-free graphs. Afterwards we were able to improve this result to (C_4, P_9)-free graphs via a simplification of our construction. We would like to thank an anonymous reviewer for pointing out this simplification as well.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 35; pp. 35:1–35:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Graph colouring has been a popular and extensively studied concept in computer science and mathematics since its introduction as a map colouring problem more than 150 years ago due to its many application areas crossing disciplinary boundaries and to its use as a benchmark problem in research into computational hardness. The corresponding decision problem, Colouring, is to decide, for a given graph G and integer k, if G admits a *k-colouring*, that is, a mapping $c : V(G) \to \{1, \ldots, k\}$ such that $c(u) \neq c(v)$ whenever $uv \in E(G)$. Unless $\mathsf{P} = \mathsf{NP}$, it is not possible to solve Colouring in polynomial time for general graphs, not even if the number of colours is limited to 3 [37]. To get a better understanding of the borderline between tractable and intractable instances of Colouring, it is natural to restrict the input to some special graph class. Hereditary graph classes, which are classes of graphs closed under vertex deletion, provide a unified framework for a large collection of well-known graph classes. It is readily seen that a graph class is hereditary if and only if it can be characterized by a (unique) set $\mathcal{H}$ of minimal forbidden induced subgraphs. Graphs with no induced subgraph isomorphic to a graph in a set $\mathcal{H}$ are called *$\mathcal{H}$-free*.

Over the years, the study of Colouring for hereditary graph classes has evolved into a deep area of research in theoretical computer science and discrete mathematics (see, for example, [6, 22, 31, 44]). One of the best-known results is the classical result of Grötschel, Lovász, and Schrijver [24], who showed that Colouring is polynomial-time solvable for perfect graphs. Faster, even linear-time, algorithms are known for subclasses of perfect graphs, such as chordal graphs, bipartite graphs, interval graphs, and comparability graphs; see for example [22]. All these classes are characterized by infinitely many minimal forbidden induced subgraphs. Král', Kratochvíl, Tuza, and Woeginger [35] initiated a systematic study into the computational complexity of Colouring restricted to hereditary graph classes characterized by a *finite* number of minimal forbidden induced subgraphs. In particular they gave a complete classification of the complexity of Colouring for the case where $\mathcal{H}$ consists of a single graph H. Their dichotomy result led to two natural directions for further research:

1. Is it possible to obtain a dichotomy for Colouring on H-free graphs if the number of colours, k, is fixed (that is, k no longer belongs to the input)?
2. Is it possible to obtain a dichotomy for Colouring on $\mathcal{H}$-free graphs if $\mathcal{H}$ has size 2?

We briefly discuss known results for both directions below and refer to [19] for a detailed survey. Let C_s and P_t denote the cycle on s vertices and path on t vertices, respectively. We start with the first question. If k is fixed, then we denote the problem by k-Colouring. It is known that for every $k \geq 3$, the k-Colouring problem on H-free graphs is NP-complete whenever H contains a cycle [16] or an induced claw [28, 36]. Therefore, only the case when H is a disjoint union of paths remains. In particular, the situation where $H = P_t$ has been thoroughly studied. On the positive side, 3-Colouring P_7-free graphs and k-Colouring P_5-free graphs for any fixed $k \geq 1$ are shown to be polynomial-time solvable [3, 26]. On the negative side, Huang [29] proved NP-completeness for ($k = 5$, $t = 6$) and for ($k = 4$, $t = 7$). The cases ($k = 3$, $t \geq 8$) and ($k = 4$, $t = 6$) remain open, although some partial results are known [9, 10].

In this paper we focus on the second question, that is, we restrict the input of Colouring to $\mathcal{H}$-free graphs for $\mathcal{H} = \{H_1, H_2\}$. For two graphs G and H, we use $G + H$ to denote the disjoint union of G and H, and we write rG to denote the disjoint union of r copies of G. As a starting point, Král', Kratochvíl, Tuza, and Woeginger [35] identified the following three main sources of NP-completeness: (i) both H_1 and H_2 contain a claw; (ii) both H_1 and H_2 contain a cycle; and (iii) both H_1 and H_2 contain an induced subgraph from the set

$\{4P_1, 2P_1 + P_2, 2P_2\}$. They also showed additional NP-completeness results by mixing the three types. Since then numerous papers [1, 7, 8, 13, 14, 25, 27, 29, 33, 35, 38, 41, 42, 43, 47] have been devoted to this problem, but despite all these efforts the complexity classification for Colouring on (H_1, H_2)-free graphs is still far from complete, and even dealing with specific pairs (H_1, H_2) may require substantial work.

One of the "mixed" results obtained in [35] is that Colouring is NP-complete for (C_s, H)-free graphs when $s \geq 5$ and $H \in \{4P_1, 2P_1 + P_2, 2P_2\}$. This, together with the well-known result that Colouring can be solved in linear time for P_4-free graphs, implies the following dichotomy.

▶ **Theorem 1** ([35]). *Let $s \geq 5$ be a fixed positive integer. Then* Colouring *for (C_s, P_t)-free graphs is polynomial-time solvable when $t \leq 4$ and* NP*-complete when $t \geq 5$.*

Theorem 1 raises the natural question: what is the complexity of Colouring on (C_s, P_t)-free graphs when $s \in \{3, 4\}$? Huang, Johnson and Paulusma [30] proved that 4-Colouring, and thus Colouring, is NP-complete for (C_3, P_{22})-free graphs, while a result of Brandstädt, Klembt and Mahfud [5] implies that Colouring is polynomial-time solvable for (C_3, P_6)-free graphs. For $s = 4$, it is only known that Colouring is polynomial-time solvable for (C_4, P_5)-free graphs [41]. This is unless we fix the number of colours: for every $k \geq 1$ and $t \geq 1$, it is known that k-Colouring is polynomial-time solvable for (C_4, P_t)-free graphs [21].

Our Results. We first show, in section 3, that Colouring is polynomial-time solvable for (C_4, P_6)-free graphs. This case was explicitly mentioned as a natural case to consider in [19]. Our result unifies several previous results on colouring (C_4, P_t)-free graphs, namely: the polynomial-time solvability of Colouring for (C_4, P_5)-free graphs [41]; the polynomial-time solvability of k-Colouring for (C_4, P_6)-free graphs for every $k \geq 1$ [21]; and the recent 3/2-approximation algorithm for Colouring for (C_4, P_6)-free graphs [18]. It was not previously known if there exists an integer t such that Colouring is NP-complete for (C_4, P_t)-free graphs. In section 4 we complement our positive result by giving an affirmative answer to this question: already the value $t = 9$ makes the problem NP-complete.

Our Methodology. The general research aim of our paper is to increase, in a systematic way, our insights in the computational hardness of Colouring and to narrow the complexity gaps between hard and easy cases. Clique-width is a well-known width parameter and having bounded clique-width is often the underlying reason for a large collection of NP-complete problems, including Colouring, to become tractable on a special graph class; this follows from results of [11, 17, 34, 45, 46]. However, the class of (C_4, P_6)-free graphs contains the class of split graphs, which may have arbitrarily large clique-width [40]. Hence, if we want to use clique-width to solve Colouring for (C_4, P_6)-free graphs, then we first need to preprocess the input graph. An *atom* is a graph with no clique cut set. In this paper we prove that (C_4, P_6)-free atoms *have* bounded clique-width. This implies a polynomial-time algorithm for Colouring on (C_4, P_6)-free graphs, as it is well known that Colouring is polynomial-time solvable on a hereditary graph class $\mathcal{G}$ if it is so on the atoms of $\mathcal{G}$ [49].

In order to prove that (C_4, P_6)-free atoms have bounded clique-width, we further develop the approach of [18] that was used to bound the chromatic number of (C_4, P_6)-free graphs as a linear function of their maximum clique size and to obtain a 3/2-approximation algorithm for Colouring for (C_4, P_6)-free graphs. The approach of [18] is based on a decomposition theorem for (C_4, P_6)-free atoms. For our purposes we derive a new variant of this decomposition theorem for so-called strong atoms, which are atoms that contain no universal

vertices and no pairs of twin vertices. Another novel element in our approach is that we show how to bound the clique-width of a graph by the clique-width of its subgraphs induced by homogeneous pairs of sets, and this will be very useful for dealing with (C_4, P_6)-free strong atoms. To apply this method, we also need to use divide-and-conquer to bound the clique-width of subgraphs induced by homogeneous pairs of sets.

2 Preliminaries

For general graph theory notation we follow [2]. Let $G = (V, E)$ be a graph. The *neighbourhood* of a vertex v, denoted by $N_G(v)$, is the set of neighbours of v. For a set $X \subseteq V(G)$, let $N_G(X) = \bigcup_{v \in X} N_G(v) \setminus X$. The *degree* of v, denoted by $d_G(v)$, is equal to $|N_G(v)|$. For $x \in V$ and $S \subseteq V$, we denote by $N_S(x)$ the set of neighbours of x that are in S, i.e., $N_S(x) = N_G(x) \cap S$. For $X, Y \subseteq V$, we say that X is *complete* (resp. *anti-complete*) to Y if every vertex in X is adjacent (resp. non-adjacent) to every vertex in Y. A vertex subset $K \subseteq V$ is a *clique cutset* if $G - K$ has more components than G and K induces a clique. A vertex is *universal* in G if it is adjacent to all other vertices. For $S \subseteq V$, the subgraph *induced* by S, is denoted by $G[S]$.

A subset $D \subseteq V$ is a *dominating set* if every vertex not in D has a neighbour in D. Let $u, v \in V$ be two distinct vertices. We say that a vertex $x \notin \{u, v\}$ *distinguishes* u and v if x is adjacent to exactly one of u and v. A set $H \subseteq V$ is a *homogeneous set* if no vertex in $V \setminus H$ can distinguish two vertices in H. A homogeneous set H is *proper* if $1 < |H| < |V|$. A graph is *prime* if it contains no proper homogeneous set. We say that u and v are *twins* if u and v are adjacent and they have the same set of neighbours in $V \setminus \{u, v\}$. Note that the binary relation of being twins is an equivalence relation on V and so V can be partitioned into equivalence classes $T_1, \ldots, T_r$ of twins. The *skeleton* of G is the subgraph induced by a set of r vertices, one from each of $T_1, \ldots, T_r$. A *blow-up* of a graph G is a graph G' obtained by replacing each vertex v of G with a clique K_v of size at least 1 such that K_v and K_u are complete in G' if u and v are adjacent in G, and anti-complete otherwise. Since each equivalence class of twins is a clique and any two equivalence classes are either complete or anti-complete, every graph is a blow-up of its skeleton.

The *clique-width* of a graph G, denoted by $cw(G)$, is the minimum number of labels required to construct G using the following four operations:

- $i(v)$: create a new graph consisting of a single vertex v with label i;
- $G_1 \oplus G_2$: take the disjoint union of two labelled graphs G_1 and G_2;
- $\eta_{i,j}$: join each vertex with label i to each vertex with label j (for $i \neq j$);
- $\rho_{i \to j}$: rename label i to j.

A *clique-width expression* for G is an algebraic expression that describes how G can be recursively constructed using these operations. A *k-expression* for G is a clique-width expression using at most k distinct labels. For instance, this is a 3-expression for the induced path on four vertices a, b, c, d:

$$\eta_{3,2}(3(d) \oplus \rho_{3 \to 2}(\rho_{2 \to 1}(\eta_{3,2}(3(c) \oplus \eta_{2,1}(2(b) \oplus 1(a)))))).$$

Clique-width is of fundamental importance in computer science since all problems expressible in monadic second-order logic using quantifiers over vertex subsets but not over edge subsets become polynomial-time solvable for graphs of bounded clique-width [11]. Although this meta-theorem does not directly apply to Colouring, a result of Kobler and Rotics [34], combined with the approximation algorithm of Oum and Seymour [45] for finding a p-expression, showed that Colouring can be added to the list of such problems.

▶ **Theorem 2** ([34]). COLOURING *can be solved in polynomial time for graphs of bounded clique-width.*

3 The Polynomial-Time Result

In this section, we shall prove that the chromatic number of any (C_4, P_6)-free graph can be found in polynomial time.

▶ **Theorem 3.** COLOURING *is polynomial-time solvable on the class of* (C_4, P_6)*-free graphs.*

A graph is called an *atom* if it contains no clique cutset. The main ingredient for proving Theorem 3 is a new structural property of (C_4, P_6)-free atoms below which asserts that (C_4, P_6)-free atoms have bounded clique-width.

▶ **Theorem 4.** *Every* (C_4, P_6)*-free atom has bounded clique-width.*

The proof of Theorem 4 is deferred to subsection 3.3.

Proof of Theorem 3 (assuming Theorem 4). Let G be a (C_4, P_6)-free graph. We find the clique decomposition of Tarjan [49] in $O(mn)$ time and this gives a binary decomposition tree T where the root of T is G and the leaves are induced subgraphs of G without clique cutsets. Tarjan [49] showed that there are at most $O(n)$ leaves and that the chromatic number of any node in T is the maximum of the chromatic numbers of its children. Therefore, determining $\chi(G)$ reduces to determining the chromatic number of atoms. Now it follows from Theorem 4 that each atom has bounded clique-width and thus the chromatic number can be found in polynomial time by Theorem 2. ◀

The remainder of the section is organized as follows. In subsection 3.1, we present the key tools on clique-width that play an important role in the proof of Theorem 4. In subsection 3.2, we list structural properties around a 5-cycle in a (C_4, P_6)-free graph that are frequently used in later proofs. We then present our main proof, the proof of Theorem 4, in subsection 3.3.

3.1 Clique-width

Let $G = (V, E)$ be a graph and H be a proper homogeneous set in G. Then $V \setminus H$ is partitioned into two subsets N and M where N is complete to H and M is anti-complete to H. Let $h \in H$ be an arbitrary vertex and $G_h = G - (H \setminus \{h\})$. We say that H and G_h are *factors* of G with respect to H. Suppose that τ is a k_1-expression for G_h using labels $1, \ldots, k_1$ and σ is a k_2-expression for H using labels $1, \ldots, k_2$. Then substituting $i(h)$ in τ with $\rho_{1\to i} \ldots \rho_{k_2\to i}\sigma$ results in a k-expression for G where $k = \max\{k_1, k_2\}$.

▶ **Lemma 5** ([12]). *The clique-width of any graph* G *is the maximum clique-width of any prime induced subgraph of* G*.*

A bipartite graph is a *chain* graph if it is $2P_2$-free. A *co-bipartite chain graph* is the complement of a bipartite chain graph. Let G be a (not necessarily bipartite) graph such that $V(G)$ is partitioned into two subsets A and B. We say that a k-expression for G is *nice* if all vertices in A end up with the same label i and all vertices in B end up with the same label j with $i \neq j$. It is well-known that any co-bipartite chain graph whose vertex set is partitioned into two cliques has a nice 4-expression.

▶ **Lemma 6** (Folklore). *There is a nice* 4*-expression for any co-bipartite chain graph.*

We now use divide-and conquer to show that a special graph class has clique-width at most 4. This plays a crucial role in our proof of the main theorem (Theorem 4).

▶ **Lemma 7.** *Let G be a C_4-free graph such that $V(G)$ is partitioned into two subsets A and B that satisfy the following conditions:*

- **(i)** *A is a clique;*
- **(ii)** *B is P_4-free;*
- **(iii)** *no vertex in A has two non-adjacent neighbours in B;*
- **(iv)** *there is no induced P_4 in G that starts with a vertex in A followed by three vertices in B.*

Then there is a nice 4-expression for G.

Proof. We use induction on $|B|$. If B contains at most one vertex, then G is a co-bipartite chain graph and the lemma follows from Theorem 6. So, we assume that B contains at least two vertices. Since B is P_4-free, either B or $\overline{B}$ is disconnected [48]. Suppose first that B is disconnected. Then B can be partitioned into two nonempty subsets B_1 and B_2 that are anti-complete to each other. Let $A_1 = N(B_1) \cap A$ and $A_2 = A \setminus A_1$. Clearly, $G[A_i \cup B_i]$ with the partition (A_i, B_i) satisfies all the conditions of the lemma for each $1 \leq i \leq 2$. Note also that, by (iii), A_1 is anti-complete to B_2 and A_2 is anti-complete to B_1. By the inductive hypothesis there is a nice 4-expression τ_i for $G[A_i \cup B_i]$ in which all vertices in A_i and B_i have labels 2 and 4, respectively. Now $\rho_{1\to 2}(\eta_{1,2}(\tau_1 \oplus \rho_{2\to 1}\tau_2))$ is a nice 4-expression for G.

Suppose now that $\overline{B}$ is disconnected. This means that B can be partitioned into two subsets B_1 and B_2 that are complete to each other. Since G is C_4-free, either B_1 or B_2 is a clique. Without loss generality, we may assume that B_1 is a clique. Moreover, we choose the partition (B_1, B_2) such that B_1 is maximal. Then every vertex in B_2 is not adjacent to some vertex in B_2 for otherwise we could have moved such a vertex to B_1. If $B_2 = \emptyset$ then G is a co-bipartite chain graph and so the lemma follows from Theorem 6. Therefore, we assume in the following that $B_1, B_2 \neq \emptyset$. Let $A_1 = N(B_1) \cap A$ and $A_2 = A \setminus A_1$. Note that A_2 is anti-complete to B_1.

We claim that $N(B_2) \cap A$ is complete to B_1. Suppose, by contradiction, that $a \in N(B_2) \cap A$ and $b_1 \in B_1$ are not adjacent. By definition, a has a neighbour $b \in B_2$. Recall that b is not adjacent to some vertex $b' \in B_2$. Now a, b, b_1, b' induces either a P_4 or a C_4, depending on whether a and b' are adjacent. This contradicts (iv) or the C_4-freeness of G. This proves the claim. Therefore, A_2 is anti-complete to B_2 and $N(B_2) \cap A = N(B_2) \cap A_1$ (see Figure 1). Consequently, $G[(A_1 \cap N(B_2)) \cup B_2]$ with the partition $(A_1 \cap N(B_2), B_2)$ satisfies all the conditions of the lemma. By the inductive hypothesis there is a nice 4-expression τ for $G[(A_1 \cap N(B_2)) \cup B_2]$ in which all vertices in $A \cap N(B_2) = A_1 \cap N(B_2)$ and B_2 have labels 2 and 4, respectively. In addition, note that $(A_1 \setminus N(B_2), B_1)$ is a co-bipartite chain graph. It then follows from Theorem 6 that there is a nice 4-expression ϵ for it in which all vertices in $A_1 \setminus N(B_2)$ and B_1 have labels 1 and 3, respectively. Then

$$\sigma = \rho_{3\to 4}(\rho_{1\to 2}(\eta_{3,4}(\eta_{2,3}(\eta_{1,2}(\epsilon \oplus \tau)))))$$

is a nice 4-expression for $G - A_2$. Let δ be a 2-expression for A_2 in which all vertices in A_2 have label 1. Then $\rho_{1\to 2}(\eta_{1,2}(\delta \oplus \sigma))$ is a nice 4-expression for G. This completes the proof. ◀

Let $G = (V, E)$ be a graph and X and Y two disjoint subsets of $V(G)$. We say that (X, Y) is a *homogeneous pair of sets* in G if no vertex in $V \setminus (X \cup Y)$ distinguishes two vertices in X or in Y. If both X and Y are cliques then (X, Y) is a *homogeneous pair of*

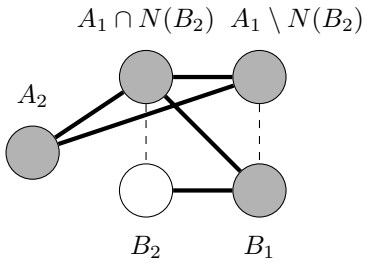


Figure 1 The case $\overline{B}$ is disconnected. Shaded circles represent cliques. A thick line between two sets represents that the two sets are complete, and a dotted line represents that the edges between the two sets can be arbitrary. Two sets are anti-complete if there is no line between them.

cliques. Note that homogeneous sets are special cases of homogeneous pair of sets where one of X and Y is empty. We establish a novel framework via existing results on clique-width for bounding the clique-width of a graph by the clique-width of its subgraphs induced by homogeneous pairs of sets.

▶ **Lemma 8.** *Let G be a graph such that $V(G)$ can be partitioned into a subset V_0 of vertices of constant size, a constant number of pairs of sets (S_i, T_i) for $1 \leq i \leq r$ and a subset V' of vertices such that*

(i) *for each $1 \leq i \leq r$, (S_i, T_i) is a homogeneous pair of sets in $G - (V_0 \cup \bigcup_{j=1}^{i-1}(S_j \cup T_j)$;*
(ii) *for each $1 \leq i \leq r$, $G[S_i \cup T_i]$ has bounded clique-width; and*
(iii) *$G[V']$ has bounded clique-width.*

Then G has bounded clique-width.

Proof. Let $G_1 = G - V_0$ and $G_{i+1} = G_i - (S_i \cup T_i)$ for $1 \leq i \leq r$. Note that $G_{r+1} = G[V']$. First of all, it follows from [39] that G has bounded clique-width if and only if G_1 has bounded clique-width. In addition, (i) says that (S_i, T_i) is a homogeneous pair of sets in G_i. Let N_i and M_i be sets of vertices in G_i that are complete to S_i and T_i, respectively. For each i we do in G_i two *bipartite complementations* on the pairs $(S_i, V(G_i) \setminus N_i)$ and $(T_i, V(G_i) \setminus M_i)$, which means that we interchange edges and non-edges between the pairs. This results in a graph G' on the same vertex set as G_1 that is the disjoint union of $G[S_i \cup T_i]$ and $G[V']$. It follows from [32] that G_1 has bounded clique-width if and only if each $G[S_i \cup T_i]$ and $G[V']$ have bounded clique-width. Now the lemma follows from our assumptions (ii) and (iii). ◀

3.2 Structure around a 5-Cycle

Let $G = (V, E)$ be a graph and H be an induced subgraph of G. We partition $V \setminus V(H)$ into subsets with respect to H as follows: for any $X \subseteq V(H)$, we denote by $S(X)$ the set of vertices in $V \setminus V(H)$ that have X as their neighbourhood among $V(H)$, i.e.,

$$S(X) = \{v \in V \setminus V(H) : N_{V(H)}(v) = X\}.$$

For $0 \leq j \leq |V(H)|$, we denote by S_j the set of vertices in $V \setminus V(H)$ that have exactly j neighbours among $V(H)$. Note that $S_j = \bigcup_{X \subseteq V(H):|X|=j} S(X)$. We say that a vertex in S_j is a *j-vertex*. Let G be a (C_4, P_6)-free graph and $C = 1, 2, 3, 4, 5$ be an induced C_5 in G. We partition $V \setminus C$ with respect to C as above. All indices below are modulo 5. Since G is C_4-free, there is no vertex in $V \setminus C$ that is adjacent to vertices i and $i + 2$ but not to vertex $i + 1$. In particular, $S(1, 3)$, S_4, etc. are empty. The following properties of $S(X)$ were proved in [25] using the fact that G is (C_4, P_6)-free.

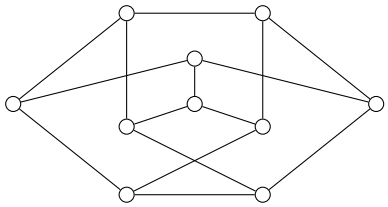


Figure 2 The Petersen graph.

(P1) $S_5 \cup S(i-1,i,i+1)$ is a clique.
(P2) $S(i)$ is complete to $S(i+2)$ and anti-complete to $S(i+1)$. Moreover, if neither $S(i)$ nor $S(i+2)$ are empty then both sets are cliques.
(P3) $S(i,i+1)$ is complete to $S(i+1,i+2)$ and anti-complete to $S(i+2,i+3)$. Moreover, if neither $S(i,i+1)$ nor $S(i+1,i+2)$ are empty then both sets are cliques.
(P4) $S(i-1,i,i+1)$ is anti-complete to $S(i+1,i+2,i+3)$.
(P5) $S(i)$ is anti-complete to $S(j,j+1)$ if $j \neq i+2$. Moreover, if a vertex in $S(i+2,i+3)$ is not anti-complete to $S(i)$ then it is universal in $S(i+2,i+3)$.
(P6) $S(i)$ is anti-complete to $S(i+1,i+2,i+3)$.
(P7) $S(i-2,i+2)$ is anti-complete to $S(i-1,i,i+1)$.
(P8) Either $S(i)$ or $S(i+1,i+2)$ is empty. By symmetry, either $S(i)$ or $S(i-1,i-2)$ is empty.
(P9) At least one of $S(i-1,i)$, $S(i,i+1)$ and $S(i+2,i-2)$ is empty.

3.3 Proof of Theorem 4

In this section, we give a proof of Theorem 4. A graph is *chordal* if it does not contain any induced cycle of length at least 4. The following structure of (C_4, P_6)-free graphs discovered by Brandstädt and Hoàng [4] is of particular importance in our proofs below.

▶ **Theorem 9** ([4]). *Let G be a (C_4, P_6)-free atom. Then the following statements hold:*
(i) *every induced C_5 is dominating;*
(ii) *if G contains an induced C_6 which is not dominating, then G is the join of a blow-up of the Petersen graph (Figure 2) and a (possibly empty) clique.*

We say that an atom is *strong* if it has no pair of twin vertices or universal vertices. Note that a pair of twin vertices and a universal vertex in a graph give rise to two special kinds of proper homogeneous sets such that one of the factors decomposed by these homogeneous sets is a clique. Therefore, removing twin vertices and universal vertices does not change the clique-width of the graph by Theorem 5. So, to prove Theorem 4 it suffices to prove the theorem for strong atoms. We follow the approach in [18]. In [18], the first and second authors showed how to derive a useful decomposition theorem for (C_4, P_6)-free atoms by eliminating a sequence F_1, C_6, F_2 and C_5 (see Figure 3 for the graphs F_1 and F_2) of induced subgraphs and then employing Dirac's classical theorem [15] on chordal graphs. Here we adopt the same strategy and show in Theorem 10–Theorem 13 below that if a (C_4, P_6)-free strong atom G contains an induced C_5 or C_6, then it has bounded clique-width. The remaining case is therefore that G is chordal and so G is a clique by Dirac's theorem [15]. Since cliques have clique-width 2, Theorem 4 follows. It turns out that we can easily prove Theorem 10 and Theorem 11 via the framework formulated in Theorem 8 using the structure of the graphs discovered in [18]. The difficulty is, however, that we have to extend the structural analysis

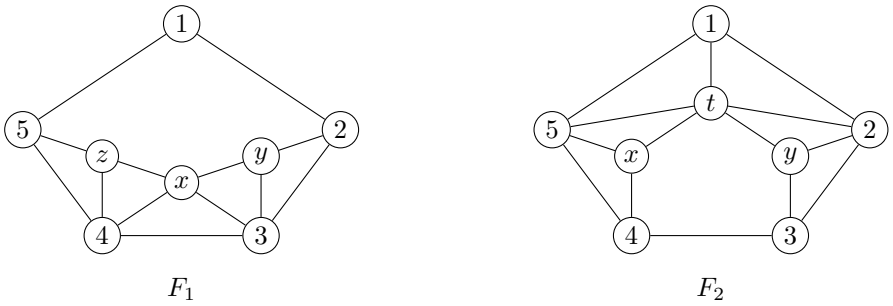


Figure 3 Two special graphs F_1 and F_2.

in [18] extensively for Theorem 12 and Theorem 13 and provide new insights on bounding the clique-width of certain special graphs using divide-and-conquer (see Theorem 7).

▶ **Lemma 10.** *If a (C_4, P_6)-free strong atom G contains an induced F_1, then G has bounded clique-width.*

▶ **Lemma 11.** *If a (C_4, F_1, P_6)-free strong atom G contains an induced C_6, then G has bounded clique-width.*

▶ **Lemma 12.** *If a (C_4, C_6, F_1, P_6)-free strong atom G contains an induced F_2, then G has bounded clique-width.*

▶ **Lemma 13.** *If a $(C_4, C_6, F_1, F_2, P_6)$-free strong atom G contains an induced C_5, then G has bounded clique-width.*

We illustrate our techniques by giving a proof of Theorem 12 below and omit the proofs of the other lemmas.

Proof of Theorem 12. Let G be a (C_4, C_6, F_1, P_6)-free strong atom that contains an induced subgraph H that is isomorphic to F_2 with $V(H) = \{1, 2, 3, 4, 5, t, x, y\}$ such that $1, 2, 3, 4, 5, 1$ induces the *underlying* 5-cycle C, and t is adjacent to 5, 1 and 2, x is adjacent to $4, 5$ and y is adjacent to 2 and 3. Moreover, t is adjacent to both x and y, see Figure 3. We partition $V(G)$ with respect to C. We choose H such that C has $|S_2|$ maximized. Note that $x \in S(4, 5)$, $y \in S(2, 3)$ and $t \in S(5, 1, 2)$.

The overall strategy is to first decompose G into a subset V_0 of constant size, constant number of homogeneous pairs of sets, and a subset V', and then finish off the proof via Theorem 8 by showing that each homogeneous pair of sets and $G[V']$ have bounded clique-width where Theorem 7 is employed.

We start with the decomposition. Since $S(2, 3)$ and $S(4, 5)$ are not empty, it follows from (P8) that $S_1 = S(2) \cup S(5)$. If both $S(2)$ and $S(5)$ are not empty, say $u \in S(2)$ and $v \in S(5)$, then $u, 2, 3, 4, 5, v$ induces either a P_6 or a C_6, depending on whether u and v are adjacent. This shows that $S_1 = S(i)$ for some $i \in \{2, 5\}$. Now we argue that $S_2 = S(2, 3) \cup S(4, 5)$. If $S(3, 4)$ contains a vertex z, then z is adjacent to x and y by (P3) but not adjacent to t by (P7). This implies that t, x, z, y induces a C_4, So, $S(3, 4) = \emptyset$. If $S(1, 2)$ contains a vertex z, then z is adjacent to y by (P3) and so $1, z, y, 3, 4, 5, 1$ induces a C_6, a contradiction. This shows that $S(1, 2) = \emptyset$. By symmetry, $S(5, 1) = \emptyset$. Therefore, $S_2 = S(2, 3) \cup S(4, 5)$. The following properties among subsets of G were proved in [18].

(a) Each vertex in $S(5, 1, 2)$ is either complete or anti-complete to S_2.
(b) $S(2, 3)$ and $S(4, 5)$ are cliques.

(c) Each vertex in $S(3,4,5) \cup S(4,5,1)$ is either complete or anti-complete to $S(4,5)$. By symmetry, each vertex in $S(1,2,3) \cup S(2,3,4)$ is either complete or anti-complete to $S(2,3)$.
(d) $S(4,5)$ is anti-complete to $S(2,3,4)$. By symmetry, $S(2,3)$ is anti-complete to $S(3,4,5)$.
(e) $S(1,2,3)$ is complete to $S(5,1,2)$. By symmetry, $S(5,1,2)$ is complete to $S(4,5,1)$.
(f) $S(4,5)$ is complete to $S(4,5,1)$. By symmetry, $S(2,3)$ is complete to $S(1,2,3)$.
(g) $S(1,2,3)$ is complete to $S(2,3,4)$. By symmetry, $S(3,4,5)$ is complete to $S(4,5,1)$.
(h) S_5 is complete to S_2.

Recall that $S_1 = S(i)$ for some $i \in \{2,5\}$. By symmetry, we may assume that $S_1 = S(5)$. Note that $S(5)$ is complete to $S(4,5,1)$ by Theorem 9 and anti-complete to $S(1,2,3)$ by (P6). It follows from (P1), (P4), (P7), (e), (f) and (g) that $S(i-1,i,i+1) \cup \{i\}$ is a homogeneous clique in G and therefore $S(i-1,i,i+1) = \emptyset$ for $i = 2,5$. Similarly, $S(4,5)$ is a homogeneous clique in G by (P7), (a)-(d), (f) and (h) and so $S(4,5) = \{x\}$. Let $T = \{t \in S(5,1,2) : t \text{ is complete to } S_2\}$.

(1) *$S(5)$ is anti-complete to $S(5,1,2) \setminus T$.*
Let $u \in S(5)$ and $t' \in S(5,1,2) \setminus T$. If u and t' are adjacent, then $u, t', 2, 3, 4, x$ induces either a P_6 or a C_6, depending on whether u and x are adjacent. ∎

By (1) and (d), $(S(5,1,2) \setminus T) \cup \{1\}$ is a homogeneous set in G and so $S(5,1,2) \setminus T = \emptyset$. In other words, $S(5,1,2)$ is complete to S_2. We now partition $S(5)$ into $X = \{v \in S(5) : v \text{ has a neighbour in } S(2,3)\}$ and $Y = S(5) \setminus X$.

(2) *X is anti-complete to $S(3,4,5)$.*
Let $v \in X$ and $s \in S(3,4,5)$ be adjacent. By the definition of X, v has a neighbour $y' \in S(2,3)$. By (d), y' is not adjacent to s and so $v, y', 3, s$ induces a C_4. ∎

(3) *X is complete to $S(5,1,2)$.*
Assume, by contradiction, that $v \in X$ and $t' \in T$ are not adjacent. By the definition of X, v has a neighbour $y' \in S(2,3)$. Since t' is adjacent to y', $v, 5, t', y'$ induces a C_4. ∎

(4) *X is anti-complete to Y.*
Suppose that $u \in X$ and $v \in Y$ are adjacent. Let $y' \in S(2,3)$ be a neighbour of u. Note that x is adjacent to neither u nor v by (P5). But now $x, 4, 3, y', u, v$ induces a P_6. ∎

(5) *X is complete to S_5.*
Suppose that $v \in X$ and $u \in S_5$ are not adjacent. Let $y' \in S(2,3)$ be a neighbour of v. By (h), y' and u are adjacent. Then $u, 5, v, y'$ induces a C_4. ∎

It follows from (P1)-(P7), (a)-(d), (f), (h) and (2)-(5) that $(X, S(2,3))$ is a homogeneous pair of sets in G.

(6) *For each connected component A of Y, each vertex in $S(5,1,2) \cup S(3,4,5)$ is either complete or anti-complete to A.*
Let A be an arbitrary component of Y. Suppose that $s \in S(5,1,2) \cup S(3,4,5)$ distinguishes an edge aa' in A, say s is adjacent to a but not adjacent to a'. we may assume by symmetry that $s \in S(5,1,2)$. Then $a', a, s, 2, 3, 4$ induces a P_6, a contradiction. ∎

(7) *Each component of Y has a neighbour in both $S(5,1,2)$ and $S(3,4,5)$.*
Suppose that a component A of Y does not have a neighbour in one of $S(5,1,2)$ and $S(3,4,5)$, say $S(5,1,2)$. Then $S_5 \cup S(3,4,5) \cup \{5\}$ is a clique cutset of G by (4). ∎

(8) *Each component of Y is a clique.*
Let A be an arbitrary component of Y. By (7), A has a neighbour $s \in S(5,1,2)$ and $r \in S(3,4,5)$. Note that s and r are not adjacent. Moreover, $\{s,r\}$ is complete to A by (6). Now (8) follows from the fact that G is C_4-free. ∎

(9) *Y is complete to S_5.*
Suppose, by contradiction, that $v \in Y$ and $u \in S_5$ are not adjacent. By (7), v has a neighbour $s \in S(5,1,2)$ and $r \in S(3,4,5)$. Then v,s,u,r induces a C_4. ∎

It follows from (P1), (h), (5) and (9) that each vertex in S_5 is a universal vertex in G and so $S_5 = \emptyset$. Let $S'(3,4,5) = \{s \in S(3,4,5) : s \text{ has a neighbour in } Y\}$ and $S''(3,4,5) = S(3,4,5) \setminus S'(3,4,5)$. Note that $S''(3,4,5)$ is anti-complete to Y. We now show further properties of Y and $S'(3,4,5)$.

(10) *$S'(3,4,5)$ is complete to $S(2,3,4)$.*
Suppose, by contradiction, that $r' \in S'(3,4,5)$ is not adjacent to $s \in S(2,3,4)$. By the definition of $S'(3,4,5)$, r has a neighbour $v \in Y$. Then $v,r,4,s,2,1$ induces a P_6. ∎

(11) *Each vertex in $S(5,1,2)$ is either complete or anti-complete to Y.*
Let $t' \in S(5,1,2)$ be an arbitrary vertex. Suppose that t' has a neighbour $u \in Y$. Let A be the component of Y containing u. Then t' is complete to A by (6). It remains to show that t' is adjacent to each vertex $u' \in Y \setminus A$. By (7), u has a neighbour $s \in S(3,4,5)$. Note that $C' = u,t,'y,3,s$ induces a C_5. Moreover, x and s are not adjacent for otherwise x,s,u,t' induces a C_4. This implies that x is adjacent only to t' on C'. On the other hand, u' is not adjacent to any of u, 3 and y. This implies that u' is adjacent to either s or t' by Theorem 9. If u' is not adjacent t', then u' is adjacent to s. This implies that $u',s,3,y,t',x$ induces a P_6 or C_6, depending on whether u' and x are adjacent. Therefore, u' is adjacent to t'. Since u' is an arbitrary vertex in $Y \setminus A$, this proves (11). ∎

(12) *$S'(3,4,5)$ is anti-complete to x.*
Suppose not. Let $s \in S'(3,4,5)$ be adjacent to x. By definition, s has a neighbour $y' \in Y$. Note that x and y' are not adjacent by (P5). By (6) and (7), y has a neighbour $t \in T = S(5,1,2)$. So, t is adjacent to x. But now s,y',t,x induces a C_4. ∎

It follows from (P1)-(P7), (d), (2), (4), (10), (11) and (12) that $(Y, S'(3,4,5))$ is a homogeneous pair of sets in G. Let $S'(5,1,2) = \{s \in S(5,1,2) : s \text{ is complete to } Y\}$. Then $S(5,1,2) \setminus S'(5,1,2)$ is anti-complete to Y by (11). It follows from (3) that both $S'(5,1,2)$ and $S(5,1,2) \setminus S'(5,1,2)$ are homogeneous cliques in G. So, $|S(5,1,2)| \leq 2$. *Now $V(G)$ is partitioned into a subset $V_0 = C \cup S(5,1,2) \cup \{x\}$ of vertices of size at most 8, two homogeneous pairs of sets $(X, S(2,3))$ and $(Y, S'(3,4,5))$, and a subset $V' = S''(3,4,5) \cup S(2,3,4)$.*

We now apply Theorem 8 to finish off the proof by showing that each of $G[X \cup S(2,3)]$, $G[Y \cup S'(3,4,5)]$, and $G[V']$ has bounded clique-width. First of all, $G[V']$ has clique-width 4 by Theorem 6. Secondly, note that no vertex in $S(1,2)$ can have two non-adjacent neighbours in X since G is C_4-free. If there is an induced $P_4 = y', x_1, x_2, x_3$ such that $y' \in S(2,3)$ and $x_i \in X$, then $x_3, x_2, x_1, y', 3, 4$ induces a P_6 in G. Now if $P = x_1, x_2, x_3, x_4$ is an induced P_4 in $G[X]$, any neighbour y_1 of x_1 is not adjacent to x_3 and x_4. But then $P \cup \{y_1\}$ contains such a labelled P_4 in $G[X \cup S(2,3)]$. Therefore, $G[X \cup S(2,3)]$ with the partition $(X, S(2,3))$ satisfies all the conditions of Theorem 7 and so has clique-width at most 4. Finally, note that each vertex in $S(3,4,5)$ can have neighbours in at most one component of Y due to (7), (11) and the fact that G is C_4-free. It then follows from (6)-(8) that $G[Y \cup S'(3,4,5)]$ with

the partition $(Y, S'(3,4,5))$ satisfies all the condition in Theorem 7 (where $A = S'(3,4,5)$ and $B = Y$) and so has clique-width at most 4. This completes our proof. ◀

We are now ready to prove our main theorem.

Proof of Theorem 4. Let G be a (C_4, P_6)-free atom. Let G' be the graph obtained from G by removing twin vertices and universal vertices. It follows from Theorem 10–Theorem 13 that if G' contains an induced C_5 or C_6, then G' has bounded clique-width. Therefore, we can assume that G' is also (C_5, C_6)-free and therefore G' is chordal. It then follows from a well-known result of Dirac [15] that G' is a clique whose clique-width is 2. Finally, $cw(G) = cw(G')$ by Theorem 5 and this completes the proof. ◀

4 The Hardness Result

A graph is a *split graph* if its vertex set can be partitioned into two disjoint sets C and I such that C is a clique and I is an independent set. The pair (C, I) is called a *split partition* of G. A split graph is *complete* if it has a *complete* split partition, that is, a partition (C, I) such that C and I are complete to each other. A *list assignment* of a graph $G = (V, E)$ is a function L that prescribes, for each $u \in V$, a finite list $L(u) \subseteq \{1, 2, \dots\}$ of colours for u. The *size* of a list assignment L is the maximum list size $|L(u)|$ over all vertices $u \in V$. A colouring c *respects* L if $c(u) \in L(u)$ for all $u \in V$. The List Colouring problem is to decide whether a given graph G has a colouring c that respects a given list assignment L. We sketch a proof of our hardness result, in which we construct a graph G' that is neither $(sP_2 + P_8)$-free nor $(sP_2 + P_4 + P_5)$-free for any $s \geq 0$. Hence, a different construction is needed for tightening our hardness result (if possible).

▶ **Theorem 14.** Colouring *is* NP*-complete for* $(C_4, 3P_3, P_3 + P_6, 2P_5, P_9)$*-free graphs.*

Proof. (Sketch) We reduce from List Colouring on complete split graphs with a list assignment of size at most 3. It is known that List Colouring is NP-complete for this graph class [20].

Let G be a complete split graph with a list assignment L of size at most 3. From (G, L) we construct an instance (G', k) of Colouring as follows. Let $k \leq 3|V(G)|$ be the size of the union of all lists $L(u)$. Let (C, I) be a complete split partition of $V(G)$. Let G' be the graph of size $O(|V(G)|k)$ obtained from G as follows. Take a clique X on k vertices $x_1, \dots, x_k$. For each $u \in V(G)$, introduce a clique Y_u of size $k - |L(u)|$ such that every vertex of Y_u is adjacent to u and to every x_i with $i \in L(u)$ (so, each vertex in every Y_u is adjacent to exactly one vertex of $V(G)$, namely vertex u). By construction, G has a colouring that respects L if and only if G' has a k-colouring. Moreover, it can be readily checked that G' is $(C_4, 3P_3, P_3 + P_6, 2P_5, P_9)$-free. ◀

5 Conclusions

We gave an almost complete dichotomy for Colouring restricted to (C_4, P_t)-free graphs and leave open only the cases when $7 \leq t \leq 8$. We believe the techniques developed in this paper could be useful for solving open questions regarding Colouring on other hereditary graph classes. The natural candidate class for a polynomial-time result of Colouring is the class of (C_4, P_7)-free graphs. However, this may require significant efforts for the following reason. Lozin and Malyshev [38] determined the complexity of Colouring for $\mathcal{H}$-free graphs for every finite set of graphs $\mathcal{H}$ consisting only of 4-vertex graphs except when

$\mathcal{H}$ is $\{K_{1,3}, 4P_1\}$, $\{K_{1,3}, 2P_1 + P_2\}$, $\{K_{1,3}, 2P_1 + P_2, 4P_1\}$ or $\{C_4, 4P_1\}$. Solving any of these open cases would be considered as a major advancement in the area. Since $(C_4, 4P_1)$-free graphs are (C_4, P_7)-free, polynomial-time solvability of Colouring on (C_4, P_7)-free graphs implies polynomial-time solvability for Colouring on $(C_4, 4P_1)$-free graphs. As a first step, we aim to apply the techniques of this paper to $(C_4, 4P_1)$-free graphs.

We recall that the complexity of Colouring on (C_s, P_t)-free graphs is known for all $s \geq 5$ and $t \geq 1$ (Theorem 1) and that the complexity of Colouring on (C_3, P_t)-free graphs is also known due to the results of [5] and [30] except if $7 \leq t \leq 21$. The class of (C_3, P_7)-free graphs is also a natural class to consider. Interestingly, every (C_3, P_7)-free graph is 5-colourable. This follows from a result of Gravier, Hoàng and Maffray [23] who proved that for any two integers $r, t \geq 1$, every (K_r, P_t)-free graph can be coloured with at most $(t-2)^{r-2}$ colours. On the other hand, 3-Colouring is polynomial-time solvable for P_7-free graphs [3]. Hence, in order to solve Colouring for (C_3, P_7)-free graphs we may instead consider 4-Colouring for (C_3, P_7)-free graphs. This problem seems also highly non-trivial.

References

1. Alexandre Blanché, Konrad K. Dabrowski, Matthew Johnson, and Daniël Paulusma. Hereditary graph classes: When the complexities of Colouring and Clique Cover coincide. *CoRR*, abs/1607.06757, 2016.
2. John Adrian Bondy and Uppaluri S.R. Murty. *Graph Theory*, volume 244 of *Springer Graduate Texts in Mathematics*. Springer, 2008.
3. Flavia Bonomo, Maria Chudnovsky, Peter Maceli, Oliver Schaudt, Maya Stein, and Mingxian Zhong. Three-coloring and list three-coloring of graphs without induced paths on seven vertices. *Combinatorica*, (in press).
4. Andreas Brandstädt and Chính T. Hoàng. On clique separators, nearly chordal graphs, and the maximum weight stable set problem. *Theoretical Computer Science*, 389:295–306, 2007.
5. Andreas Brandstädt, Tilo Klembt, and Suhail Mahfud. P_6- and triangle-free graphs revisited: structure and bounded clique-width. *Discrete Mathematics and Theoretical Computer Science*, 8:173–188, 2006.
6. Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*, volume 3 of *SIAM Monographs on Discrete Mathematics and Applications*. Philadelphia, 1999.
7. Hajo Broersma, Petr A. Golovach, Daniël Paulusma, and Jian Song. Determining the chromatic number of triangle-free $2P_3$-free graphs in polynomial time. *Theoretical Computer Science*, 423:1–10, 2012.
8. Kathie Cameron and Chính T. Hoàng. Solving the clique cover problem on (bull,C_4)-free graphs. *CoRR*, abs/1704.00316, 2017.
9. Maria Chudnovsky, Peter Maceli, Juraj Stacho, and Mingxian Zhong. 4-Coloring p_6-free graphs with no induced 5-cycles. *Journal of Graph Theory*, 84:262–285, 2017.
10. Maria Chudnovsky and Juraj Stacho. 3-colorable subclasses of P_8-free graphs. *Manuscript*, 2017.
11. Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33:125–150, 2000.
12. Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101:77–114, 2000.
13. Konrad K. Dabrowski, François Dross, and Daniël Paulusma. Colouring diamond-free graphs. *Journal of Computer and System Sciences*, 89:410–431, 2017.

14 Konrad K. Dabrowski, Petr A. Golovach, and Daniël Paulusma. Colouring of graphs with Ramsey-type forbidden subgraphs. *Theoretical Computer Science*, 522:34–43, 2014.

15 Gabriel A. Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25:71–76, 1961.

16 Thomas Emden-Weinert, Stefan Hougardy, and Bernd Kreuter. Uniquely colourable graphs and the hardness of colouring graphs of large girth. *Combinatorics, Probability and Computing*, 7:375–386, 1998.

17 Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. *Proc. WG 2001, LNCS*, 2204:117–128, 2001.

18 Serge Gaspers and Shenwei Huang. Linearly χ-bounding (P_6, C_4)-free graphs. *Proc. WG 2017, LNCS*, 10520:263–274, 2017.

19 Petr A. Golovach, Matthew Johnson, Daniël Paulusma, and Jian Song. A survey on the computational complexity of coloring graphs with forbidden subgraphs. *Journal of Graph Theory*, 84:331–363, 2017.

20 Petr A. Golovach and Daniël Paulusma. List coloring in the absence of two subgraphs. *Discrete Applied Mathematics*, 166:123–130, 2014.

21 Petr A. Golovach, Daniël Paulusma, and Jian Song. Coloring graphs without short cycles and long induced paths. *Discrete Applied Mathematics*, 167:107–120, 2014.

22 Martin C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, volume 57 of *Annals of Discrete Mathematics*. North-Holland Publishing Co., 2004.

23 Sylvain Gravier, Chính T. Hoàng, and Frédéric Maffray. Coloring the hypergraph of maximal cliques of a graph with no long path. *Discrete Mathematics*, 272:285–290, 2003.

24 Martin Grötschel, László Lovász, and Alexander Schrijver. Polynomial algorithms for perfect graphs. *Annals of Discrete Mathematics*, 21:325–356, 1984.

25 Pavol Hell and Shenwei Huang. Complexity of coloring graphs without paths and cycles. *Discrete Applied Mathematics*, 216, Part 1:211–232, 2017.

26 Chính T. Hoàng, Marcin Kamiński, Vadim V. Lozin, Joe Sawada, and Xiao Shu. Deciding k-colorability of P_5-free graphs in polynomial time. *Algorithmica*, 57:74–81, 2010.

27 Chính T. Hoàng and D. Adam Lazzarato. Polynomial-time algorithms for minimum weighted colorings of $(P_5, \overline{P_5})$-free graphs and similar graph classes. *Discrete Applied Mathematics*, 186:106–111, 2015.

28 Ian Holyer. The NP-Completeness of edge-coloring. *SIAM Journal on Computing*, 10:718–720, 1981.

29 Shenwei Huang. Improved complexity results on k-coloring P_t-free graphs. *European Journal of Combinatorics*, 51:336–346, 2016.

30 Shenwei Huang, Matthew Johnson, and Daniël Paulusma. Narrowing the complexity gap for colouring (C_s, P_t)-free graphs. *The Computer Journal*, 58:3074–3088, 2015.

31 Tommy R. Jensen and Bjarne Toft. *Graph Coloring Problems*. Wiley Interscience, 1995.

32 Marcin Kamiński, Vadim V. Lozin, and Martin Milanič. Recent developments on graphs of bounded clique-width. *Discrete Applied Mathematics*, 157:2747–2761, 2009.

33 Thiyagarajan Karthick, Frédéric Maffray, and Lucas Pastor. Polynomial cases for the vertex coloring problem. *CoRR*, abs/1709.07712, 2017.

34 Daniel Kobler and Udi Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Applied Mathematics*, 126:197–221, 2003.

35 Daniel Král', Jan Kratochvíl, Zsolt Tuza, and Gerhard J. Woeginger. Complexity of coloring graphs without forbidden induced subgraphs. *Proc. WG 2001, LNCS*, 2204:254–262, 2001.

36 Daniel Leven and Zvi Galil. NP completeness of finding the chromatic index of regular graphs. *Journal of Algorithms*, 4:35–44, 1983.

37 László Lovász. Coverings and coloring of hypergraphs. *Congressus Numerantium*, VIII:3–12, 1973.

38 Vadim V. Lozin and Dmitriy S. Malyshev. Vertex coloring of graphs with few obstructions. *Discrete Applied Mathematics*, 216, Part 1:273–280, 2017.

39 Vadim V. Lozin and Dieter Rautenbach. On the band-, tree-, and clique-width of graphs with bounded vertex degree. *SIAM Journal on Discrete Mathematics*, 18:195–206, 2004.

40 Johann A. Makowsky and Udi Rotics. On the clique-width of graphs with few P_4's. *International Journal of Foundations of Computer Science*, 10:329–348, 1999.

41 Dmitriy S. Malyshev. The coloring problem for classes with two small obstructions. *Optimization Letters*, 8:2261–2270, 2014.

42 Dmitriy S. Malyshev. Two cases of polynomial-time solvability for the coloring problem. *Journal of Combinatorial Optimization*, 31:833–845, 2016.

43 Dmitriy S. Malyshev and O. O. Lobanova. Two complexity results for the vertex coloring problem. *Discrete Applied Mathematics*, 219:158–166, 2017.

44 Michael Molloy and Bruce Reed. *Graph Colouring and the Probabilistic Method*. Springer-Verlag Berlin Heidelberg, 2002.

45 Sang-Il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96:514–528, 2006.

46 Michaël Rao. MSOL partitioning problems on graphs of bounded treewidth and clique-width. *Theoretical Computer Science*, 377:260–267, 2007.

47 David Schindl. Some new hereditary classes where graph coloring remains NP-hard. *Discrete Mathematics*, 295:197–202, 2005.

48 D. Seinsche. On a property of n-colorable graphs. *Journal of Combinatorial Theory, Series B*, 16:191–193, 1974.

49 Robert E. Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55:221–232, 1985.

Optimal Dislocation with Persistent Errors in Subquadratic Time

Barbara Geissmann
Department of Computer Science, ETH Zürich, Switzerland

Stefano Leucci
Department of Computer Science, ETH Zürich, Switzerland

Chih-Hung Liu
Department of Computer Science, ETH Zürich, Switzerland

Paolo Penna
Department of Computer Science, ETH Zürich, Switzerland

Abstract

We study the problem of sorting N elements in presence of *persistent* errors in comparisons: In this classical model, each comparison between two elements is wrong independently with some probability p, but repeating the same comparison gives always the same result. The best known algorithms for this problem have running time $O(N^2)$ and achieve an optimal maximum dislocation of $O(\log N)$ for constant error probability. Note that no algorithm can achieve dislocation $o(\log N)$, regardless of its running time.

In this work we present the first *subquadratic* time algorithm with optimal maximum dislocation: Our algorithm runs in $\widetilde{O}(N^{3/2})$ time and guarantees $O(\log N)$ maximum dislocation with high probability. Though the first version of our algorithm is randomized, it can be derandomized by extracting the necessary random bits from the results of the comparisons (errors).

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Sorting, Recurrent Comparison Errors, Maximum Dislocation

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.36

1 Introduction

We study the problem of *sorting* N distinct elements under *recurrent* random comparison *errors*. In this classical model, each comparison is wrong with some fixed (small) probability p, and correct with probability $1 - p$. The probability of errors are independent over all possible pairs of elements, but errors are recurrent: Repeating the same comparison several times is useless since the result is always the same, i.e., always wrong or always correct.

Because of errors, different sorting algorithms can have different guarantees to output a "nearly sorted" sequence. To measure the quality of an output sequence in terms of sortedness, a common way is to consider the *dislocation* of an element, which is the difference between its position in the output and its position in the correctly sorted sequence. In particular, one can consider the *maximum dislocation* of any element in the permutation or the *total dislocation* of a permutation, i.e., the sum of the dislocations of all n elements. Of course, the running *time* is also an important criteria for evaluating sorting algorithms.

Regarding the *maximum dislocation* and the *running time*, in the recurrent random comparison errors, this is the state of the art:

- Several algorithms [3, 12, 9] guarantee *maximum dislocation* $O(\log N)$ with high probability, though their *running time* is *quadratic* or even *larger* (see Table 1).

Table 1 The running time of previous algorithms which guarantee $O(\log N)$ maximum dislocation (with high probability) and our result. The constant $c(p)$ in [3] depends on both the error probability p, and the success probability of the algorithm, and it is typically quite large. The algorithms in [12, 9] have different guarantees on the total dislocation.

	Braverman and Mossel [3]	Klein et al [12], Geissmann et al [9]	**This work**
Time	$O(N^{3+c(p)})$	$O(N^2)$	$\widetilde{O}(N^{3/2})$

- No algorithm (even randomized) can achieve *maximum dislocation* $o(\log N)$ with high probability, *regardless of its running time* [9].

This suggests naturally the following question:

> *Is there any algorithm with* ***subquadratic running time*** *which achieves* ***optimal maximum dislocation*** $O(\log n)$ *with high probability?*

In this paper we give an affirmative answer to this question.

1.1 Our contribution

We present the first *subquadratic time* algorithm with *optimal maximum dislocation*, namely, an algorithm that runs in $\widetilde{O}(N^{3/2})$ time and returns a sequence of maximum dislocation $O(\log N)$ with high probability (see Table 1). The latter is optimal because, in the model with *persistent* errors, no algorithm (even randomized) can achieve maximum dislocation $o(\log N)$ with high probability [9]. Intuitively speaking, our algorithm (Recursive Window Sort) first picks a random permutation and then performs a number of deterministic operations which use the algorithm in [9] as a subroutine. All recursive steps are *deterministic* and they consist of an algorithm that approximately sorts an input sequence whenever it is *well shuffled* and the errors are *well spread*. The latter condition holds with high probability in the error model we consider, and the starting random permutation serves to have a well shuffled input. The correctness of Recursive Window Sort combines a technical condition that the algorithm in [9] guarantees, combined with an intermediate "merge step" which works well on well shuffled inputs (see Section 2 for an high level description of the algorithm and the main ideas).

Though our first algorithm is *randomized*, it can be "derandomized" in the following sense. By using the results of the comparison errors, the algorithm itself can generate the necessary (almost) random bits to be used in the computation. Note that this is far from trivial for two reasons: (i) The outcome of the comparisons are also used during the computation and (ii) The result of a comparison may tell something about the result of another comparison. Our second major contribution is a *deterministic* algorithm (Derandomized Recursive Window Sort) which still runs in $\widetilde{O}(N^{3/2})$ and that returns a sequence of maximum dislocation $O(\log N)$ with high probability (over the random comparison errors).

Connections with prior work

The algorithm by Braverman and Mossel [3] constructs the maximum likehood permutation, whose computation requires a rather large (though polynomial) running time. Their method in fact uses only $O(N \log N)$ comparisons and is applicable to any $p < 1/2$, while the faster algorithms by Klein et al [12] and Geissmann et al [9] work for p smaller than some absolute small constant (e.g., in [9] $p < 1/16$).

Ajtai et al [1] provide algorithms using *subquadratic* time (and number of comparisons) when errors occur only between elements whose difference is at most some fixed threshold. Damaschke [6] gives also a *subquadratic* time algorithm, by assuming that at most k errors occur. The algorithm returns a sequence up to $O(k)$ inversions and it is based on finding a solution for the feedback arc set (FAST) problem. Coppersmith and Rurda [5] provide a simple algorithm 5-approximation for the weighted FAST problem, if the weights satisfy probability constraints.

An easier error model is the one with *non-recurrent* errors, meaning that the same comparison can be *repeated* and the errors are independent with some probability $p < 1/2$. For this model, Feige et al. [7] gave a sorting algorithm running in $O(N \log(N/q))$ steps, where $1 - q$ is the success probability of the algorithm. Alonso et al. [2] and Hadjicostas and Lakshamanan [11] studied the classical Quicksort and recursive Mergesort algorithms, respectively. Sorting by repeatedly performing random swaps results in Markovian processes which have been studied by Geissmann et al [8, 10].

Finally, computing with errors is often considered in the framework of a two-person game called *Rényi-Ulam Game* (see Pelc's survey [14] and Cicalese's monograph [4]).

1.2 Preliminaries

In this section, we describe the key features of the Window Sort algorithm [9] which will be used a a subroutine of our main algorithm (see Algorithm 1). To this end, we first introduce some notation used throughout the paper.

We consider the problem of sorting N distinct integers which, under the error model considered here, is equivalent to sort a sequence containing the integers $\{1, 2, \ldots, N\}$. For any sequence S, and any element x in the sequence, we define its *rank* as the number of elements smaller than x in S, i.e., $rank(x, S) \stackrel{\triangle}{=} |\{y \in S |\ y < x\}|$. Note that this gives the correct position of x (its rank plus 1) in the correctly sorted sequence, and it only depends on the elements in S (not in the sequence order). In the following we will use $\kappa \geq 1$ to denote a global constant that only depends on the error probability p. For ease of presentation, we assume that $p < \frac{1}{32}$, although our algorithm can be adapted to work for $p < \frac{1}{16}$ (which is a condition needed to successfully run Window Sort). We say that a comparison between an (unordered) pair of elements x, y, with $x < y$, is an *error* if x is (incorrectly) reported to be *larger than* y.

▶ **Definition 1.** We define $ERRORS(x, w, S)$ as the set of errors among the comparisons between element x and every other element y in S with $rank(y, S) \in [rank(x, S) - 4w, rank(x, S) + 4w]$.

▶ **Definition 2.** For a set of elements S, we say that the comparison errors are *well spread* iff, for all $x \in S$ and for all w such that $\kappa \log |S| \leq w \leq n$, we have $|ERRORS(x, w, S)| \leq w/4$.

▶ **Definition 3** (Success)**.** We say that $\langle S, W \rangle$, where S is a sequence and W a window size, satisfies the Success condition if

1. The maximum dislocation of S is at most W;
2. The comparison errors in S are *well spread*.

This condition guarantees that the output of Window Sort will have maximum dislocation $O(\log |S|)$, where the initial window size determines its running time:

▶ **Lemma 4** ([9])**.** Window Sort *on a sequence S with a starting window size W returns a sequence having maximum dislocation at most $\kappa \log n$ in $O(|S| \cdot W)$ time whenever $\langle S, W \rangle$*

Algorithm 1: WINDOW SORT (on a sequence S of n distinct elements and initial window size W).

Initialization: The initial window size is $w = W$. Each element x has two variables $wins(x)$ and $computed\_rank(x)$ which are set to zero.

repeat

1. **foreach** x at position $l = 1, 2, 3, \ldots, n$ in S **do**
   foreach y whose position in S is in $[l - 2w, l - 1]$ *or* in $[l + 1, l + 2w]$ **do**
   if $x > y$ **then**
   $wins(x) = wins(x) + 1$
   $computed\_rank(x) = \max\{l - 2w, 0\} + wins(x)$
2. Place the elements into S' ordered by non-decreasing $computed\_rank$, break ties arbitrarily.
3. Set all $wins$ to zero, $S = S'$, and $w = w/2$.

until $w < 1$;

satisfies the SUCCESS *condition. Moreover, the expected total dislocation of the returned sequence is* $O(n)$.

▶ **Lemma 5.** *For any sequence S of n elements chosen independently of the errors, the probability (over the comparison errors) that errors are well spread is at least* $1 - 1/n^8$.

2 Warm up

In this section we informally describe some of the ideas used in our algorithm. As a warm up, we consider a simplified (non-optimized) version which is described in Figure 1 and consists of the following steps:

1. Start with a *random permutation* S of the input sequence and split this sequence S into β blocks of the same size.[1]
2. Run WINDOW SORT on each block B_i to obtain a sequence S_i.
3. Combine all the sequences S_i together into a sequence S' as follows: The first element in each S_i will be placed (in arbitrary order) in one of the first β positions of S', the second element in each S_i will be placed in a position between $\beta + 1$ and 2β in S', and so on.
4. Run WINDOW SORT on this new sequence S'.

At this point two observations are in place. First, we did not specify yet some parameters, namely, the number β of blocks (and thus their size N/β), nor the initial window size when we call WINDOW SORT. Both these parameters need to be chosen carefully in order to achieve the desired performance and, in our more complex scheme, they will vary at every recursive call. Second, the initial step where we pick a random permutation of the input (the elements to be sorted) can be implemented more efficiently by distributing directly these elements into the desired number of blocks.

Saving in the running time

One intuition why this scheme should be faster than WINDOW SORT, is that in the first part this algorithm is called on smaller blocks and the running time is thus $\widetilde{O}(N^2/\beta)$, since each

[1] For the sake of simplicity, here and in the rest of the paper, we assume that $|S|$ is a multiple of the block size.

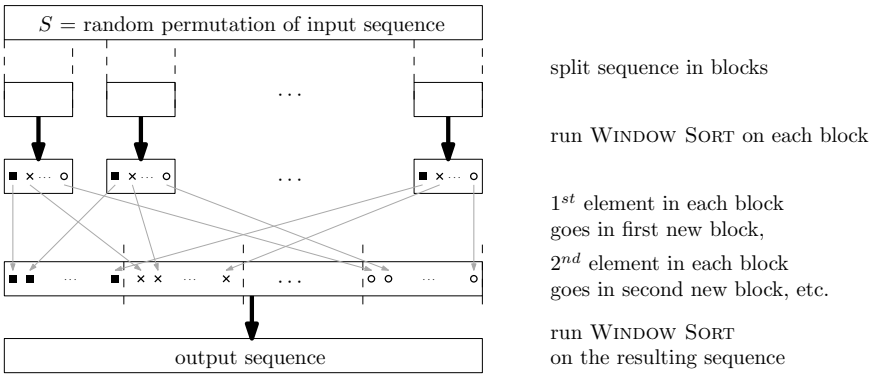


Figure 1 The one-level recursion scheme.

block takes $\widetilde{O}((N/\beta)^2)$ time. The last call to Window Sort can be fast if S' has already a bounded maximum dislocation, since in this case we can start the algorithm with a small initial window size.

Need for initial random permutation

To see why we perform this initial step, consider the version in which we start with the sorted sequence, that is, $S = \langle 1, 2, \ldots, N\rangle$. We claim that, in this case, it is very likely that the sequence S' obtained after recombining the blocks has *large dislocation*. Indeed, suppose all the calls to Window Sort sort perfectly each block. Then, $S_1 = \langle 1, 2, \ldots, N/\beta\rangle$ and $S_2 = \langle N/\beta+1, \ldots, 2N/\beta\rangle$, causing element 2 to be placed in S' in position at least $N/\beta+1$.

Correctness argument

In order to get the desired bound on the maximum dislocation, we essentially need to show that every call to Window Sort on some *subsequence* of elements will be "successful", that is, the output will have a bounded maximum dislocation. Note that these calls are *not* independent since the results of the comparisons (which depend on the errors) determine the sequence S' in the last call to Window Sort. It turns out, that for any fixed subset of elements and any fixed window size that is large enough (i.e., logarithmic in N), this property holds with *high probability* (w.r.t. N). Moreover, the input sequence S' in the last call of Window Sort involves *all* elements, while the other β calls for the blocks involve randomly chosen subsets of elements (independent of the errors). As all these subsets are polynomially many (we choose β accordingly below), all these calls to Window Sort succeed with high probability too (union bound).

3 The algorithm

We now describe the full version of our algorithm, which we call Recursive Window Sort. Intuitively, our algorithm is a recursive version of the scheme described in Section 2 (see Figure 1), where the only randomized part is the initial shuffling of the input sequence, which is performed only once. We refer to this random permutation of the input sequence as S. All recursive steps are *deterministic* and they consist of an algorithm that approximately sorts an input sequence whenever it is *well shuffled* and the errors are *well spread*.

We next describe the recursive steps of Recursive Window Sort. We denote by N the total number of elements to sort. The behavior of our recursive algorithm varies according

Algorithm 2: RECURSIVE WINDOW SORT (on N distinct elements).

Let S be a random permutation the N input elements
Run RECURSIVE STEP on S (with initial depth $d = 0$)
return the resulting sequence

Algorithm 3: RECURSIVE STEP (on a sequence S of n_d distinct elements at depth d).

Initialization: the maximum depth is $h = \log N$, the values β_d and W_d are chosen as in (1)
if $d = h$ **then**
  Run WINDOW SORT on $S' = S$ with window size n_d
  return the resulting sequence
else
  Partition S into $b_d \triangleq \frac{n_d}{\beta_d}$ blocks $B_1, B_2, \ldots, B_{b_d}$ each containing β_d elements
  foreach block B_i **do**
    Run RECURSIVE STEP on B_i with depth $d+1$ to obtain $B'_i = \langle b'_{i,1}, b'_{i,2}, \ldots, b'_{i,\beta_d} \rangle$
  foreach $j = 1, 2, \ldots, \beta_d$ **do**
    $B''_j = \langle b'_{1,j}, b'_{2,j}, \ldots, b'_{b_d,j} \rangle$
  Let $S' = \langle s'_1, s'_2, \ldots, s'_{n_d} \rangle = \langle B''_1, B''_2, \ldots, B''_{\beta_d} \rangle$
  Run WINDOW SORT on S' with window size W_d
  return the resulting sequence

to the current depth of recursion. The maximum depth of the recursion is $h = \log N$.[2] In general, a recursive step at depth d sorts an input sequence S of n_d elements[3], by splitting S into blocks of size β_d and recursing on these blocks. Then, it recombines the elements from the blocks in a zip fastener fashion into a single sequence S', and runs WINDOW SORT on S' with window size W_d. We formally describe such a recursive step in Algorithm 3 and RECURSIVE WINDOW SORT in Algorithm 2.

In order to optimize the running time, we shall set the parameters as follows:

$$\beta_d \triangleq n_d^{1-\frac{1}{2^{h-d+1}-1}} \qquad \text{and} \qquad W_d \triangleq 4\kappa \frac{n_d}{\sqrt{\beta_d}} \log N. \tag{1}$$

3.1 Running time

We begin by providing an upper bound on the running time of RECURSIVE WINDOW SORT.

▶ **Lemma 6.** *The overall running time of* RECURSIVE WINDOW SORT *is* $\widetilde{O}(N^{\frac{3}{2}})$.

Proof. Recall that the running time of WINDOW SORT on an instance of n elements with starting window size W is upper bounded by $c'nW \log n$ for some constant $c' \geq 1$ (Lemma 4). Consider an execution of our algorithm whose depth d defines an *index* $i = h - d$. We now prove by induction on i that its running time T_i is upper bounded by $c(i+1)n_d^{1+2^i/(2^{i+1}-1)} \log N$, where $c = 4\kappa c'$. (Notice that c is a global constant that does not depend on i.)

[2] To avoid being distracted by rounding, we assume that h is an integer.
[3] Here n_d is a function of both N and d.

If $i = 0$ then $d = h$ and the running time coincides with the one of Window Sort, i.e., it is less than $c'n_d^2 \leq cn_d^{1+2^0/(2^1-1)}$. This proves the base case.

For $i > 0$ the overall running time is bounded by the sum of: (i) the time required to perform $b_d = \frac{n_d}{\beta_d}$ recursive calls with depth $d+1$ (i.e., having index is $h-(d+1) = (h-d)-1 = i-1$), on instances of size $\beta_d = n_{d+1}$, and (ii) the time required to run Window Sort on an instance of size n_d and initial window size $W_d = 4\kappa\frac{n_d}{\sqrt{\beta_d}}\log N$. By inductive hypothesis, each of the recursive calls requires time $T_{i-1} \leq c\,i\,n_{d+1}^{1+\frac{2^{i-1}}{(2^i-1)}}\log N = ci\beta_d^{1+\frac{2^{i-1}}{2^i-1}}\log N$. Thus

$$T_i \leq \frac{n_d}{\beta_d}\cdot ci\beta_d^{1+\frac{2^{i-1}}{2^i-1}}\log N + 4\kappa c'\frac{n_d^2}{\sqrt{\beta_d}}\log N = c\left(in_d\beta_d^{\frac{2^{i-1}}{2^i-1}} + n_d^{\frac{3}{2}+\frac{1}{2^{i+2}-2}}\right)\log N$$
$$= c\left(in_d^{1+\frac{2^i}{2^{i+1}-1}} + n_d^{1+\frac{2^i}{2^{i+1}-1}}\right)\log N = c(i+1)n_d^{1+\frac{2^i}{2^{i+1}-1}}\cdot\log N.$$

This completes the proof of the inductive step. By setting $i = h$, and for a sufficiently large N, we obtain $T_h \leq c(1+\log N)N^{1+\frac{N}{2N-1}}\cdot\log N \leq 2c(\log N)^2N^{\frac{3}{2}+\frac{1}{4N-2}}$. Since $N^{\frac{1}{4N-2}} = 2^{\frac{\log N}{4N-2}} = O(1)$ we conclude that $T_h = O(N^{3/2}\log^2 N)$. ◀

3.2 Correctness

Here we will formally prove the correctness of Recursive Window Sort. To this aim, we shall first give a sufficient condition for which, if all executions at depth $d+1$ return sequences of dislocation $\kappa\log N$, then also the execution at depth d returns a sequence of dislocation at most $\kappa\log N$.

▶ **Definition 7** (Good Blocks)**.** We say that an execution of Recursive Step at depth $d < h$ has Good Blocks if the sequence S to which we apply the recursion satisfies the following condition: For any element x in S, $\left|L_x - \frac{\beta_d}{n_d}G_x\right| \leq 2\sqrt{\beta_d}\log N$, for $G_x = rank(x,S)$ and $L_x = rank(x,B_j)$, where B_j is the block of length $\beta_d = n_{d+1}$ containing x.

Note that the input of each execution (recursive call) of Recursive Step is a fixed subset of elements of the initial sequence which *does not depend on the comparison errors.*

▶ **Lemma 8.** *Consider an execution of* Recursive Step *at depth $d < h$ and suppose that the following conditions hold:*

1. *The execution has* Good Blocks *(Definition 7);*
2. *All the executions at depth $d+1$ return a sequence with maximum dislocation $\kappa\log N$;*
3. *The comparison errors are well spread.*

Then, the considered execution returns a sequence with maximum dislocation $\kappa\log N$.

Proof. By hypothesis 3 together with Lemma 4, it suffices to show that the sequence S' obtained before invoking Window Sort has maximum dislocation at most $W_d = 4\kappa\frac{n_d}{\sqrt{\beta_d}}\log N$.[4] Consider an element $x \in S$, let B_j be the block containing x, and let $\widetilde{L}_x$ be the number of elements preceding x in B'_j (its position in B'_j minus 1). By the hypothesis on

[4] Notice that if the errors in S are well spread, then they are also well spread in S' since the order of the elements is irrelevant in Definition 2.

the executions at depth $d+1$, we know that $|\widetilde{L}_x - L_x| \leq \kappa \log N$ and, since the considered execution has Good Blocks, we can use triangle inequality to write

$$\left|\widetilde{L}_x - \frac{\beta_d}{n_d} G_x\right| \leq 2\sqrt{\beta_d}\log N + \kappa \log N \leq 3\kappa\sqrt{\beta_d}\log N.$$

Let $\widetilde{G}_x$ be the number of elements preceding x in S' (its position minus 1), so that $|\widetilde{G}_x - G_x|$ is the dislocation of x in S'. By definition of S' we have

$$\widetilde{G}_x \geq \frac{n_d}{\beta_d}\widetilde{L}_x \geq G_x - \frac{3\kappa n_d}{\sqrt{\beta_d}}\log N,$$

and similarly

$$\widetilde{G}_x \leq \frac{n_d}{\beta_d}\widetilde{L}_x + \frac{n_d}{\beta_d} \leq G_x + \frac{3\kappa n_d}{\sqrt{\beta_d}}\log N + \frac{n_d}{\beta_d} \leq G_x + \frac{4\kappa n_d}{\sqrt{\beta_d}}\log N\,.$$

Therefore we have shown that $\left|\widetilde{G}_x - G_x\right| \leq \frac{4\kappa n_d}{\sqrt{\beta_d}}\log N$, which concludes the proof. ◀

▶ **Lemma 9.** *The probability that all executions of* Recursive Step *at depth $d < h$ have (jointly)* Good Blocks *is at least $1 - \frac{1}{N^2}$.*

Proof. Fix an element x and a depth $d < h$. Let S and B_j be the sequence of size n_d and its block of size $b_d = n_{d+1}$ containing x. (Both S and B_j are random variables as they depend on the initial random permutation of all N elements.) Since S is a subsequence of a random permutation we can study the distribution of L_x by considering the following:

1. After the random permutation of the N input elements is chosen (thus S and B_j are determined), we randomly permute the elements of S again apart from x (which stays in its position in S and in B_j);
2. We view the previous item as the following experiment. An urn contains G_x black balls (elements smaller than x) and $n_d - G_x - 1$ white balls (elements bigger than x). Out of these $n_d - 1$ balls, choose $\beta_d - 1$ at random and consider the number of chosen black balls (the local rank L_x).

It is well known that, permuting a subsequence of a randomly chosen permutation, gives again a randomly chosen permutation. Therefore the modification of Item 1 is equivalent to the original algorithm. A random permutation of the elements in S determines which of them fall into B_j and thus Item 1 is equivalent to Item 2. The number of chosen black balls is the local rank L_x of x. We hence have that L_x is distributed as an *hypergeometric* random variable of parameters $n_d - 1$, G_x, and $\beta_d - 1$ and we can use the following tail bound [15]:

$$\Pr(|L_x - \mathbb{E}[L_x]| \geq t(\beta_d - 1)) \leq 2e^{-2t^2(\beta_d - 1)}\,,$$

where $\mathbb{E}[L_x] = \frac{\beta_d - 1}{n_d - 1}G_x$. By choosing $t = 2\sqrt{\frac{\log N}{\beta_d - 1}}$, we obtain

$$\begin{aligned}\Pr(|L_x - \frac{\beta_d}{n_d}G_x| \geq 3\sqrt{\beta_d}\log N) &\leq \Pr(|L_x - \frac{\beta_d - 1}{n_d - 1}G_x| \geq 2\sqrt{\beta_d}\log N)\\ &\leq \Pr(|L_x - \mathbb{E}[L_x]| \geq 2\sqrt{\beta_d}\log N) \leq \Pr(|L_x - \mathbb{E}[L_x]| \geq 2\sqrt{\beta_d - 1}\log N) \leq \frac{2}{N^4}.\end{aligned}$$

Notice that the overall number of elements x for which the above condition must hold is upper bounded by $N \log N$. Indeed, there are $h = \log N$ recursion levels and each level defines a partition of the N elements into blocks (i.e., the total number of elements at each level is N). By the union bound, the probability that all the executions are good is at least $1 - (N \log N)\frac{2}{N^4} > \frac{1}{N^2}$. ◀

The following two lemmas allow us to use Lemma 9 in a recursive fashion.

▶ **Lemma 10.** *For every $d = 0, \ldots, h$, it holds that $n_d \geq N^{\frac{1}{2}}$.*

▶ **Lemma 11.** *The errors of all the sequences S' are (jointly) well spread with probability at least $1 - \frac{1}{N^2}$.*

We are now ready to prove the final theorem of this section.

▶ **Theorem 12.** Recursive Window Sort *returns a sequence with maximum dislocation $\kappa \log N$ with probability at least $1 - \frac{1}{N^2}$. Moreover, its running time is $\widetilde{O}(N^{\frac{3}{2}})$ and the expected total dislocation of the returned sequence is $O(n)$.*

Proof. We assume that (i) all the recursive executions of Recursive Step have Good Blocks and that (ii) all the sequences S' used as input for Window Sort have well spread errors. By Lemma 9 and Lemma 11 this happens with probability at least $1 - \frac{2}{N^2}$).

We prove the following claim by induction on $i = h - d$: all the executions of Recursive Step at depth d return a sequence having maximum dislocation $\kappa \log N$.

If $i = 0$, then $d = h$. Consider any execution of Recursive Step at depth h and let S be its input. Notice that Window Sort is invoked on S with window size $n_d = |S|$, meaning that both conditions for Definition 3 are met and hence, by Lemma 4, Window Sort returns a sequence having maximum dislocation $\kappa \log n_d \leq \kappa \log N$.

If $i > 0$, then $d < h$ and we once again focus on any single execution of Recursive Step at depth d having input S. By inductive hypothesis all the executions at depth $d + 1$ returned a sequence having maximum dislocation $\kappa \log N$. This, combined with our assumptions, allows us to invoke Lemma 8 which proves the first claim.
To conclude the proof, notice that the running time is bounded by Lemma 6 and that, by Lemma 4, the sequence returned by the execution of Window Sort at depth $d = 0$ has expected total dislocation $O(n)$. ◀

4 Derandomization

Recursive Window Sort requires as input a random permutation of the N elements. In this section, we show how to derandomize the algorithm. In particular, we show how to generate "*almost random*" bits from the outcome of element comparisons, which can be thought as as biased coins tosses. The derandomized Recursive Window Sort is then as follows: We extract a (random) subset of elements and use them to generate random bits. Then, we use these bits to generate a random permutation of the remaining elements, which allows us to invoke Recursive Step on this permutation. Finally, we reinsert the extracted elements into the approximately sorted sequence, so that the maximum dislocation remains $O(\log N)$. Notice that the sequence returned by Recursive Step (indirectly) depends on the set of extracted elements though the results of their comparisons. We circumvent this problem by providing an algorithm that is able to reinsert a *single* element in *any* sequence having dislocation $O(\log N)$ as long as errors are well spread. We then show how this algorithm can be used to reinsert all the extracted elements without any asymptotic increase in the dislocation. For any two elements x and y we write $x \widetilde{<} y$ (resp. $x \widetilde{>} y$) to denote the fact that x *compared* smaller (resp. larger) than y.

4.1 (Re-)Inserting one element

The first key ingredient is an algorithm which reinserts an element in a sequence of n elements of maximum dislocation $O(\log n)$ so that this bound on the dislocation is maintained (up to a multiplicative constant depending on p).

Algorithm 4: InsertPosition (on a sequence $S = \langle s_0, \ldots, s_{n-1} \rangle$ of n distinct elements and on an element x not in S).

// Compute a "penalty" function for each possible position.

1. For every index $i = 0, \ldots, n$:
   a. Let $S_{i-} \triangleq \{s_{i-c_2 \log n}, \ldots, s_{i-1}\}$ and $S_{i+} \triangleq \{s_i, \ldots, s_{i+c_2 \log n - 1}\}$.
   b. Let $penalty_i(x, S) \triangleq defeats(x, S_{i-}) + wins(x, S_{i+})$.

// Return the position of minimal penalty.

2. Return any index $i^* \in \arg\min_{i=0,\ldots,n} penalty_i(x, S)$.

▶ **Definition 13** (single insertion)**.** The single insertion problem is defined as follows. We are given an arbitrary sequence[5] of n distinct elements, $S = \langle s_0, \ldots, s_{n-1} \rangle$, whose maximum dislocation is at most $c_1 \log n$, for some $c_1 \geq 1$, and another element x distinct from all these elements. The goal is to insert x in a position i^* which still guarantee $c_2 \log n$ maximum dislocation, for $c_2 := \frac{7c_1}{p}$. That is, the sequence $S' = \langle s_0, \ldots, s_{i^*-1}, x, s_{i^*}, \ldots, s_{n-1} \rangle$ has maximum dislocation at most $c_2 \log n$.

In the following, we consider these two quantities:

$$wins(x, Y) \triangleq |\{y \in Y \,:\, x \mathrel{\widetilde{>}} y\}| \qquad \text{and} \qquad defeats(x, Y) \triangleq |\{y \in Y \,:\, x \mathrel{\widetilde{<}} y\}| \,,$$

where x is an arbitrary element and Y an arbitrary subset of elements. Algorithm InsertPosition (see Algorithm 4 above) solves the single insertion problem with high probability:

▶ **Theorem 14.** *Let S be a sequence of n elements having maximum dislocation $c_1 \log n$. With probability at least $1 - \frac{3}{n^2}$ algorithm* InsertPosition *returns an index i^* such that the sequence $S' = \langle s_0, \ldots, s_{i^*-1}, x, s_{i^*}, \ldots, s_{n-1} \rangle$ has maximum dislocation at most $\frac{7c_1}{p} \log n$.*

Intuitively, the proof of this result is based on the following two facts:
1. When i is *away* from the true (correct) rank of x in S, there is a *large* penalty (Lemma 15);
2. When i is *equal* to the true (correct) rank of x in S, the penalty is *small* (Lemma 16).

▶ **Lemma 15.** *If $|i - r| \geq c_2 \log n$ then $penalty_i(x, S) \geq \frac{1-p}{2}(c_2 - 2c_1) \log n$ with probability at least $1 - \frac{2}{n^{13}}$.*

▶ **Lemma 16.** *$penalty_r(x, S) \leq \frac{1-p}{2}(c_2 - 2c_1) \log n$ with probability at least $1 - \frac{2}{n^2}$.*

Proof of Theorem 14. By Lemma 16 and by union bound on Lemma 15, we conclude that with probability at least $1 - \frac{3}{n^2}$, $penalty_r(x, S) < penalty_i(x, S)$ for every i with $|i - r| \geq c_2 \log n$. In this case, the algorithm returns a index i^*, such that $|i^* - r| < c_2 \log n$. Furthermore, the dislocation of each element between i^* and r in S changes by at most 1, and the dislocation of the other elements is unchanged. ◀

4.2 Generating almost random bits

The result $r \in \{\widetilde{<}, \widetilde{>}\}$ of comparison between two distinct elements $x, y \in S$ can be seen as a biased coin if we label its faces with $0 \triangleq \widetilde{<}$ and $1 \triangleq \widetilde{>}$: Since the comparison fails

[5] Note that S can be adversarial and can also be chosen as a function of the comparison results, of the true order of the elements, and of x.

with probability p, but we do not know the correct answer, the coin is biased towards one of its faces, i.e., either it lands on 0 with probability $1-p$ and on 1 with probability p or vice-versa. Moreover, the coin can be tossed at most once, since errors (or lack of thereof) are persistent. Consider now a collection $\mathcal{C}$ of coins whose faces are labeled 0 and 1 and let $\chi_C \in \{0,1\}$ denote the outcome of the coin flip involving coin $C \in \mathcal{C}$. For any subset $C = \{C_1, C_2, \dots\} \subseteq \mathcal{C}$ we compute the exclusive or the results $\chi(C) \triangleq \chi_{C_1} \oplus \chi_{C_2} \oplus \chi_{C_3} \oplus \cdots$ (where $\chi(C) = 0$ if $C = \emptyset$).

The next lemma shows that we can generate an almost random bits with a sufficiently large number of biased coin tosses (comparisons):

▶ **Lemma 17.** *For any choice of the coin biases, and any subset $C = \{C_1, C_2, \dots,\} \subseteq \mathcal{C}$ such that $|C| = \Omega(\log N)$, $\frac{1}{2} - \frac{1}{N^4} \leq P(\chi(C) = 0) \leq \frac{1}{2} + \frac{1}{N^4}$. (For a suitable hidden constant that depends on p).*

Notice that the above lemma holds for any choice of the coin biases (i.e., regardless of true order between the compared elements), therefore we can write the following

▶ **Corollary 18.** *For any collection $\{C^{(1)}, C^{(2)}, \dots, C^{(\eta)}\}$ of pairwise disjoint subset of $\mathcal{C}$, each of size $O(\log n)$, and any $r \in \{0,1\}^\eta$ we have that*

$$\left(\frac{1}{2} - \frac{1}{N^4}\right)^\eta \leq P\left((\chi(C^{(1)}, \dots, \chi(C^{(\eta)})) = r\right) \leq \left(\frac{1}{2} + \frac{1}{N^4}\right)^\eta.$$

Finally, we show that we are able to generate random integers in an interval that closely resemble a discrete uniform distribution.

▶ **Lemma 19.** *Let $\ell \leq N$. It is possible to generate a number z in $0, \dots, \ell-1$ using $O(\log^2 N)$ comparison results. With probability at most $\frac{1}{N^3}$, z will be a spurious result and we say that the* fail *event happens. If the fail event does not happen, then z is uniformly distributed in $0, \dots, \ell-1$.*

4.3 Derandomized Iterated Windowsort

We are now in a position to describe our deterministic algorithm Derandomized Recursive Window Sort (see Algorithm 5) and its analysis. We have already seen above how to perform and analyze most of the algorithm's steps. We will now give proofs for reinserting many elements at the same time in Step 7, and then present our main theorem for Derandomized Recursive Window Sort. For the rest of this section we let $c_3 = \frac{7\kappa}{p}$. Moreover, we will say that Derandomized Recursive Window Sort *fails* if the fail event of Lemma 19 happens at least once during the execution of the algorithm. The following two lemmas bound the dislocation of the sequence $S^{(4)}$ obtained by reinserting the elements in R after Recursive Window Sort is invoked.

▶ **Lemma 20.** *Suppose* Derandomized Recursive Window Sort *does not fail. Then, with probability at least $1 - \frac{1}{N^2}$, all the sets $R_i = \{r \in R : i \leq rank(r, S^{(1)}) \leq i + 2c_3 \log N\}$, for $0 \leq i < N$, contain at most 6 elements each.*

Proof. If there exists a set R_i that contains 7 or more elements, then there exists a corresponding set $S_j \subseteq S^{(0)}$ that satisfies: (i) $|S_j| \leq 2c_3 \log N + 8$, and (ii) $|R_i \cap S_j| \geq 7$.[6]

[6] Indeed, it suffices to choose $S_j = \{x \in S^{(0)} : j \leq rank(x, S^{(0)}) \leq j + 2c_3 \log_n + 7\}$, where $j = i + |\{r \in R\colon rank(r, S^{(1)}) < i\}|$.

Algorithm 5: DERANDOMIZED RECURSIVE WINDOW SORT (on a sequence $S = \langle s_0, \ldots, s_{n-1} \rangle$ of n distinct elements).

// Next step generates $\Omega(N)$ comparisons outcomes.

1. Let s_0 be the first element in S. Compare s_0 to every element in $S^{(0)} = S \setminus \{s_0\}$
   // This requires $O(\log^4 N)$ comparison outcomes.
2. Choose a set R of $\log^3 n$ (distinct) random elements from $S^{(0)}$ (see Lemma 19).
   // Next step generates $\Omega(N \log^3 N)$ comparison outcomes.
3. Compare each element in R with each element in $S^{(1)} = S^{(0)} \setminus R$
   // This requires $O(N \log^2 N)$ comparison outcomes (using, e.g., Fisher–Yates shuffle [13]).
4. Obtain a random permutation $S^{(2)}$ of the elements in $S^{(1)}$ (see Lemma 19).
5. Invoke RECURSIVE STEP on $S^{(2)}$ with initial depth 0 to obtain sequence $S^{(3)}$
6. For each element $x \in R$, compute its position i_x^* in $S^{(3)}$ using Algorithm 4.
7. Insert (simultaneously) each $x \in R$ in position i_x^* of $S^{(3)}$ to obtain $S^{(4)}$ (break ties arbitrarily).
8. Insert s_0 in $S^{(4)}$ using Algorithm 4 to obtain $S^{(5)}$. Return $S^{(5)}$.

We show that the probability that any single set S_i exists is at most $\frac{1}{n^3}$, so that the claim will immediately follow by using the union bound on the (at most N) values of i.

Notice that, since R is a random subset of elements of $S^{(0)}$, the probability that 7 or more elements from R belong to S_i can be upper bounded by the probability of success of the following experiment: An urn contains $|S^{(0)}| = N - 1$ balls, $|R|$ of which are black; we draw $\eta = |S_i| = 2c_3 \log N + 8$ balls without replacement and we succeed if the number X of drawn black balls is 7 or more.

Since X is distributed as an hypergeometric random variable of parameters $N-1$, $|R|$, and η, we have (for sufficiently large values of N):

$$\begin{aligned}
\Pr(X \geq 7) &= \sum_{j=9}^{\eta} \frac{\binom{|R|}{j}\binom{N-|R|-1}{\eta-j}}{\binom{N-1}{\eta}} \leq \sum_{j=7}^{\eta} |R|^j \cdot \frac{\binom{N-1}{\eta-j}}{\binom{N-1}{\eta}} \\
&= \sum_{j=7}^{\eta} |R|^j \cdot \frac{(N-1)!}{(\eta-j)!(N-1-\eta+j)!} \cdot \frac{\eta!(N-1-\eta)!}{(N-1)!} \\
&= \sum_{j=7}^{\eta} |R|^j \cdot \frac{\eta!}{(\eta-j)!} \cdot \frac{(N-1-\eta)!}{(N-1-\eta+j)!} \leq \sum_{j=7}^{\eta} \left(|R| \frac{\eta}{N-\eta}\right)^j \\
&< \sum_{j=7}^{\eta} N^{-j/2} \leq \int_{x=6}^{\infty} N^{-x/2} \mathrm{d}x = \frac{2}{N^3 \ln N} < \frac{1}{N^3}
\end{aligned}$$

◀

▶ **Lemma 21.** *Suppose* DERANDOMIZED RECURSIVE WINDOW SORT *does not fail. With probability at least* $1 - \frac{3}{N^2}$*: (i) the maximum dislocation of* $S^{(4)}$ *is at most* $c_3 \log N + 6$ *and (ii) the dislocations of an the element yin*$S^{(3)}$ *increases by at most* 6 *in* $S^{(4)}$*.*

All the previous lemmas together allow us to state the main result of this section.

▶ **Theorem 22.** *Algorithm 5 is a deterministic algorithm that returns, in* $\widetilde{O}(N^{\frac{3}{2}})$ *time, a sequence with maximum dislocation* $O(\log N)$ *and total dislocation* $O(n)$ *with probability at least* $1 - \frac{1}{N}$*.*

References

1. Miklós Ajtai, Vitaly Feldman, Avinatan Hassidim, and Jelani Nelson. Sorting and selection with imprecise comparisons. *ACM Transactions on Algorithms*, 12(2):19, 2016.
2. Laurent Alonso, Philippe Chassaing, Florent Gillet, Svante Janson, Edward M Reingold, and René Schott. Quicksort with unreliable comparisons: a probabilistic analysis. *Combinatorics, Probability and Computing*, 13(4-5):419–449, 2004.
3. Mark Braverman and Elchanan Mossel. Noisy Sorting Without Resampling. In *Proceedings of the 19th Annual Symposium on Discrete Algorithms*, pages 268–276, 2008. `arXiv:0707.1051`.
4. Ferdinando Cicalese. *Fault-Tolerant Search Algorithms - Reliable Computation with Unreliable Information*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2013.
5. Don Coppersmith, Lisa Fleischer, and Atri Rudra. Ordering by weighted number of wins gives a good ranking for weighted tournaments. *ACM Trans. Algorithms*, 6(3):55:1–55:13, 2010. `doi:10.1145/1798596.1798608`.
6. Peter Damaschke. The solution space of sorting with recurring comparison faults. In *Combinatorial Algorithms - 27th International Workshop, IWOCA 2016, Helsinki, Finland, August 17-19, 2016, Proceedings*, pages 397–408, 2016.
7. Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, 1994. `doi:10.1137/S0097539791195877`.
8. Tomas Gavenciak, Barbara Geissmann, and Johannes Lengler. Sorting by swaps with noisy comparisons. In Peter A. N. Bosman, editor, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017*, pages 1375–1382. ACM, 2017. `doi:10.1145/3071178.3071242`.
9. Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Sorting with recurrent comparison errors. *Proc of ISAAC 2017*, 2017. To appear. And *ArXiv e-prints arXiv:1709.07249, 2017*.
10. Barbara Geissmann and Paolo Penna. Sort well with energy-constrained comparisons. *ArXiv e-prints arXiv:1610.09223*, 2016.
11. Petros Hadjicostas and KB Lakshmanan. Recursive merge sort with erroneous comparisons. *Discrete Applied Mathematics*, 159(14):1398–1417, 2011.
12. Rolf Klein, Rainer Penninger, Christian Sohler, and David P. Woodruff. Tolerant Algorithms. In *Proceedings of the19th Annual European Symposium on Algorithm*, pages 736—-747, 2011.
13. Donald Ervin Knuth. *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley, 1998. URL: `http://www.worldcat.org/oclc/312898417`.
14. Andrzej Pelc. Searching games with errors - fifty years of coping with liars. *Theor. Comput. Sci.*, 270(1-2):71–109, 2002.
15. Matthew Skala. Hypergeometric tail inequalities: ending the insanity. *ArXiv e-prints arXiv:1311.5939*, 2013.

An Improved Bound for Random Binary Search Trees with Concurrent Insertions

George Giakkoupis
INRIA, Rennes, France
george.giakkoupis@inria.fr

Philipp Woelfel
University of Calgary, Canada
woelfel@cpsc.ucalgary.ca

Abstract

Recently, Aspnes and Ruppert (DISC 2016) defined the following simple random experiment to determine the impact of concurrency on the performance of binary search trees: n randomly permuted keys arrive one at a time. When a new key arrives, it is first placed into a buffer of size c. Whenever the buffer is full, or when all keys have arrived, an adversary chooses one key from the buffer and inserts it into the binary search tree.

The ability of the adversary to choose the next key to insert among c buffered keys, models a distributed system, where up to c processes try to insert keys concurrently. Aspnes and Ruppert showed that the expected average depth of nodes in the resulting tree is $O(\log n + c)$ for a *comparison-based* adversary, which can only take the relative order of arrived keys into account. We generalize and strengthen this result. In particular, we allow an adversary that knows the actual values of all keys that have arrived, and show that the resulting expected average node depth is $D_{avg}(n) + O(c)$, where $D_{avg}(n) = 2\ln n - \Theta(1)$ is the expected average node depth of a random tree obtained in the standard unbuffered version of this experiment. Extending the bound by Aspnes and Ruppert to this stronger adversary model answers one of their open questions.

2012 ACM Subject Classification Mathematics of computing → Trees, Theory of computation → Data structures design and analysis, Theory of computation → Sorting and searching

Keywords and phrases Random Binary Search Tree, Buffer, Average Depth, Concurrent Data Structures

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.37

Funding This research was undertaken, in part, thanks to funding from the ANR Project NDFusion (ANR-16-TERC-0007), the ANR Project PAMELA (ANR-16-CE23-0016-01), the Canada Research Chairs program, and the Discovery Grants program of the Natural Sciences and Engineering Research Council of Canada (NSERC).

Acknowledgements The authors are grateful to Eric Ruppert for his comments on a draft of the proof presented here. Eric also pointed out that our earlier, much simpler proof attempt was indeed too simple to be true.

1 Introduction

Consider the following random procedure, in which distinct keys from a totally ordered universe are inserted into an (internal) binary search tree (BST). Multiple keys arrive in random order, and whenever a key arrives, it is first placed into a buffer of fixed size c. Upon each key arrival, an adversary may remove one of the keys from the buffer, and insert it into

the BST. (The adversary may also decide not to insert a key, if the buffer is not full.) We are interested in performance affecting properties of the resulting BST, such as its height or average node depth.

This random experiment has recently been studied by Aspnes and Ruppert [1], to understand how concurrency influences the performance of search trees in a distributed setting: In an asynchronous shared memory system, multiple processes may concurrently try to insert keys into a BST data structure.

The order in which concurrent insertions succeed can depend on unpredictable system factors. For example, cache effects or memory locality on NUMA architectures can heavily influence the speed with which synchronization primitives, and thus insertions take effect. Events such as context switches can delay the operations by some processes, while other processes that may have locked parts of the data structure may be able to execute several operations in rapid succession. Therefore, the order in which concurrent data structure operations succeed can be arbitrary, and is usually analyzed under worst-case assumptions. For randomized algorithms or random inputs, an *adversary* is used to model how this order is influenced by random events. In analogy to the worst-case analysis of sequential algorithms, it is natural to make pessimistic adversary assumptions, i.e., consider the worst conceivable influence that random events can have on the operation order.

A buffered search tree with buffer size c as described above, models such a distributed scenario, where up to c processes insert random keys concurrently. At any point, there are up to c insert operations pending (corresponding to the keys in the buffer), and the adversary decides which of those insert operations succeeds first.

Aspnes and Ruppert [1] analyzed the buffered search tree for a *comparison based* adversary, which can base its decisions only on the relative order of the keys that have arrived so far. They showed that for a random arrival sequence of n distinct keys, the resulting BST has an expected average node depth of $O(c + \log n)$, and that this bound is asymptotically tight. They also proved that there is a comparison based adversary that can achieve an expected height of $\Omega\big(c \cdot \log(n/c)\big)$.

Note that the height of a BST corresponds to the worst-case search time, and the average node depth corresponds to the average-case search time for successful searches.

Contrary to the distributed case (buffered tree), in the sequential setting the expected height and the expected average node depth are known up to constant additive terms. For example, the expected average node depth of a BST obtained by inserting keys $\{1, \ldots, n\}$ in random order is [6]

$$D_{avg}(n) = 2(1 + 1/n)H_n - 2, \text{ where } \ln(n+1) < H_n < (\ln n) + 1.$$

It seems appropriate to also analyze height and average depth as precisely as possible for a distributed setting.

The adversary considered by Aspnes and Ruppert [1] does not fully reflect a worst-case scenario, as it can make decisions only based on the relative order of elements, but cannot distinguish their absolute values. It is not clear whether adversarial decisions based on, for example, the pairwise differences between consecutive keys in the ordered list of buffered elements, can lead to a significantly increased average node depth. Consequently, Aspnes and Ruppert suggested that "it might be interesting to see whether even stronger malicious adversaries could force the height or average depth of random trees to grow higher by using the actual values of he keys" [1]. They suggested an example for a buffer size of $c = 3$, where keys are drawn uniformly at random from the interval $[0, 1]$: If the first three keys chosen are 0.03, 0.45, and 0.54, then a good strategy for an adversary that can see absolute values

would be to insert 0.03 as the root. On the other hand, if the initial three keys in the buffer were 0.46, 0.55, and 0.97, the adversary would be better off by inserting 0.97, first.

Results. We provide a negative answer to the question posed by Aspnes and Ruppert [1], whether an adversary can achieve a significantly increase in the average node depth by using knowledge about the values of keys in the buffer. At the same time, we refine Aspnes and Ruppert's upper bound of $O(\log n+c)$ on the expected average node depth, to $D_{avg}(n)+O(c)$. As a result, a buffer of size $c = o(\log n)$ does not negatively affect the constant factor in the expected average node depth.

We consider a strong adversary, which can at any time base its next insertion decision on the exact values of all keys that have arrived so far. In a distributed system, where the buffer corresponds to pending concurrent insert operations, this translates to the strongest reasonable adversary, namely one which can base its decisions on all past random events, but not future ones.

Consider an arrival sequence of n arbitrary distinct keys chosen from a totally ordered universe and permuted randomly. Let $\Delta(n, c)$ denote the expected average node depth obtained for a buffered BST given the strong adversary, if the buffer has size c. Note that $c = 1$ corresponds to the unbuffered case, as the adversary has to move a key from the buffer whenever the buffer is full. Hence, $\Delta(n, 1) = D_{avg}(n)$ is the expected average node depth in the standard sequential case. We will prove the following result.

▶ **Theorem 1.** *$\Delta(n, c) = \Delta(n, 1) + O(c)$, for any $c \in \{2, \dots, n\}$.*

Note that the result of Aspnes and Ruppert [1] states that $\Delta'(n, c) = O(\Delta'(n, 1) + c)$, where Δ' is defined as Δ but for the weaker comparison based adversary.

Related Work. Binary search trees are among the most fundamental data structures, and their properties have been studied extensively. It has been known for a long time that the expected average depth of nodes for trees obtained by inserting a random permutation of n distinct elements is $O(\log n)$ [2, 10]. Many additional details, such as higher moments, are known about the distribution of node depths [6]. Other parameters of random BSTs have also been determined precisely. For example, a sequence of works [9, 3, 8] determined the expected height as $(4.311\dots) \cdot \log n - (1.953\dots) \cdot \log\log n + O(1)$.

The buffered search tree scenario considered in this paper was introduced by Aspnes and Ruppert [1]. They describe it in terms of a card game, where a player takes c cards from a shuffled deck of n cards, then chooses one of the cards in her hand to insert into the BST, and replaces that card with a new one from the deck. This continues until the deck has been depleted, upon which the player inserts all remaining cards in her hand into the tree. The authors point out that a complementary approach is the smoothed analysis of the expected BST height [7].

Concurrent Data Structure Context. To understand the context of this work, it is helpful to know how efficient concurrent data structures, such as binary search trees, may be implemented. There are several techniques to avoid conflicts, when multiple processes access a search tree (or other data structure) concurrently. The simplest one is to use "coarse grained locking", where a mutual exclusion algorithm (lock) protects the entire data structure. This way, only one process can access the data structure at a time, while other processes have to wait.

But such lock-based solutions are inefficient, and not fault tolerant. The "Read-Copy-Update" technique is an example of a more efficient and fault tolerant solution: The address of the root of the tree is stored in a Compare-And-Swap (CAS) object C. This is a standard shared memory primitive that supports a $C.\texttt{CAS}(old, new)$ operation, which atomically changes the value of C to new, provided its current value is old. To insert a node v, a process first creates a copy of the search path from the root to the node u of which v will become a child. Then the process executes a $C.\texttt{CAS}(old, new)$ operation, where old is the address of the original root, and new is the address of the root on the search path copy. This CAS only succeeds, if no other insertion took effect (i.e., no other CAS succeeded) in the meantime. If it fails, the process repeats its insertion attempt. This method is lock-free, guaranteeing progress even if processes crash. If the tree only supports insertions and no other update operations (such as deletions), then more efficient implementations based on fine grained synchronization are possible (e.g., using one CAS object or lock for each child pointer).

There are several other common solutions, e.g., based on transactional memory. But most of these solutions have in common that the speed of an insertion attempt may depend on the length of the search path. Since "fast" attempts have a higher chance of being successful, it is reasonable to assume that the order of concurrent insertions depends on the values of the involved keys.

2 Analysis

Our analysis works for a slightly stronger adversary, which in addition to the keys in the buffer, also knows the set (but not the order) of all future keys to arrive. In other words, the adversary knows at any point the ranks of the keys that have arrived so far, with respect to the set of all keys in the complete arrival sequence. In that setting we can assume w.l.o.g. that the key arrival sequence is a random permutation over $\{1, \ldots, n\}$. The first $c-1$ of the keys are stored in a buffer. For each following key k that arrives, first k is inserted to the buffer, and then one of the c keys in the buffer is removed and inserted into the tree. (It is obvious that the adversary can gain no advantage by inserting a key from the buffer "early", i.e., when the buffer is not full and more keys are going to arrive.) After all keys have arrived, the c keys remaining in the buffer are inserted into the tree as well.

Proof Overview. We use the fact that the average node depth in any tree is equal to the average number of descendants of all nodes in the tree. Let $k_1, \ldots, k_n$ denote the keys in the order in which they arrive (so, $k_1, \ldots, k_n$ is a random permutation of $\{1, \ldots, n\}$). We bound the expected number of descendants of each key k_i in the final tree. For $i < c$ we use the trivial bound of n on the number of descendants of k_i. Next we focus on the case $i \geq c$. We bound the expected number of descendants of k_i in the final tree, given the current tree and the set of keys in the buffer just before k_i arrives; at that time, the value of k_i is not yet determined, so it is equally likely to be any of the remaining keys. We observe that the number of descendants of k_i in the final tree is maximized if key k_i is inserted to the tree as soon as it arrives, rather than stored in the buffer and inserted at a later time. So, it suffices to bound the expected number of descendants k_i would have, if it were inserted immediately. This expected value depends only on the set of the $i-1$ keys that arrived before k_i, and the subset of them that have been inserted into the tree; in particular, it does not depend on the order in which keys arrived or the order in which keys were inserted into the tree. From this bound on the conditional expectation on the number of descendants of k_i, given the $i-1$ keys that arrived before k_i and the $i-c$ keys already in the tree, we obtain a bound

on the corresponding unconditional expectation, by letting the first $i-1$ keys be random, and assuming that an arbitrary (worst-case) $(i-c)$-subset of them has been inserted into the tree. From this bound for each $i \geq c$ (and the trivial bound of n for $i < c$), and from the linearity of expectation, we obtain a bound on the expectation of the average number of descendants of all nodes.

Detailed Proof. We now present the detailed proof of Theorem 1. As already mentioned, $k_1, \ldots, k_n$ is a random permutation of $\{1, \ldots, n\}$ denoting the order in which keys arrive. We assume w.l.o.g. that no key is inserted into the tree, unless the buffer is full or all keys have arrived. I.e., the insertions evolve in rounds as follows: In round $j \in \{1, \ldots, c-1\}$, key k_j is added to the buffer. In round $j \in \{c, \ldots, n\}$, first key k_j is added to the buffer, and then some key ℓ_j is removed from the buffer and inserted into the search tree. Finally, in each round $j \in \{n+1, \ldots, n+c-1\}$, one key ℓ_j is removed from the buffer and inserted into the tree.

For $i \in \{0, \ldots, n+c-1\}$, let X_i be the set of keys that have arrived by the end of round i, and let Y_i be the set of keys that the tree contains at the end of round i ($X_0 = Y_0 = \emptyset$). Then $X_i = \{k_1, \ldots, k_i\}$ for $i \in \{0, \ldots, n\}$, and $X_i = \{1, \ldots, n\}$ for $i > n$. Moreover, $Y_i = \emptyset$ for $i \in \{1, \ldots, c-1\}$, and $Y_i = \{\ell_c, \ldots, \ell_i\}$ for $i \in \{c, \ldots, n+c-1\}$. At the end of round i, the set of keys in the buffer is $X_i \setminus Y_i$. This set contains i elements at the end of round $i \in \{0, \ldots, c-1\}$, $c-1$ elements at the end of round $i \in \{c, \ldots, n\}$, and $n+c-i-1$ elements at the end of round $i \in \{n+1, \ldots, n+c-1\}$. Hence,

$$|X_i \setminus Y_i| = \min\{i, c-1, n+c-i-1\}. \tag{1}$$

Let $x_i^1, \ldots, x_i^i$ denote the elements of X_i ordered by increasing key values. For convenience we also define $x_i^0 = 0$ and $x_i^{i+1} = n+1$. Further, let $Y_i' = Y_i \cup \{0, n+1\}$.

We are interested in bounding the number of descendants of each key in the final tree. For $i \in \{0, \ldots, n-1\}$, define the random variable

$$\delta_i^c = \min\big\{y \in Y_i' \mid y > k_{i+1}\big\} - \max\big\{y \in Y_i' \mid y < k_{i+1}\big\}.$$

As we will show below in the proof of Claim 2, the number of descendants of k_{i+1} in the search tree with buffer size c is at most $\delta_i^c - 2$. Moreover, for $c = 1$, the number of descendants is exactly $\delta_i^1 - 2$. Since the average node depth of a search tree equals the average number of descendants of each node, we can upper bound the expected average node depth as the expected average of all values $\delta_i^c - 2$. And for $c = 1$, these two quantities match.

▶ **Claim 2.**

(a) $\Delta(n,1) = \dfrac{1}{n} \cdot \displaystyle\sum_{i=0}^{n-1} \mathbf{E}[\delta_i^1] - 2$; *and*

(b) $\Delta(n,c) \leq \dfrac{1}{n} \cdot \displaystyle\sum_{i=0}^{n-1} \mathbf{E}[\delta_i^c] - 2$, *for any* $c \geq 2$.

Proof. For $i \in \{0, \ldots, n-1\}$ let $desc(i)$ denote the set of descendants of key k_i in the final search tree, and let $depth(j)$ denote the depth of node k_j in the final tree. Then the average node depth of the final tree is

$$\frac{1}{n} \cdot \sum_{j=1}^{n} depth(j) = \frac{1}{n} \cdot \sum_{j=1}^{n} \big|\{i \in \{1, \ldots, n\} : k_j \in desc(i)\big| = \frac{1}{n} \cdot \sum_{j=1}^{n} |desc(j)|.$$

We will now show for any $i \in \{0, \dots, n-1\}$ that $|desc(i+1)| \leq \delta_i^c - 2$, and moreover, for a buffer of size $c = 1$, $|desc(i+1)| = \delta_i^1 - 2$. The claim follows immediately from that.

Key k_{i+1} arrives in the buffer in round $i+1$. Suppose it is moved into the tree in round $j+1 \geq i+1$. Then at that point the tree contains the elements in Y_j. When that happens, the largest element in the tree that is smaller than k_{i+1} (if it exists) is $a_j = \max\{y \in Y_j \mid y < k_{i+1}\}$, and the smallest element in the tree that is larger than k_{i+1} (if it exists) is $b_j = \min\{y \in Y_j \mid y > k_{i+1}\}$. The search path to the position where k_{i+1} gets inserted goes through a_j and b_j. Hence, in the final tree, the search path to any descendant of k_{i+1} also goes through those two nodes. Moreover, any key that is larger than a_j and smaller than b_j will follow the search path to k_{i+1} and become a descendant of k_{i+1}. Define $a_j = 0$ respectively $b_j = n+1$ if Y_j contains no element smaller respectively larger than k_{i+1}. Then we obtain $desc(i+1) = \{a_j + 1, \dots, b_j - 1\} \setminus \{k_{i+1}\}$, and thus

$$|desc(i+1)| = b_j - a_j - 2.$$

Now observe that if the buffer has size $c = 1$, then $j = i$, because key k_i gets inserted into the tree in the same round as it arrives. Since $a_i = \max\{y \in Y_i' \mid y < k_{i+1}\}$ and $b_i = \min\{y \in Y_i' \mid y > k_{i+1}\}$ (recall that $Y_i' = Y_i \cup \{0, n+1\}$) we obtain $\delta_i^1 = b_i - a_i$, and so $|desc(i+1)| = \delta_i^1 - 2$.

For $c > 1$ we may have $j > i$, but in any case $Y_i \subseteq Y_j$. Hence, $a_i \leq a_j$ and $b_i \geq b_j$, and so we obtain $\delta_i^c = b_i - a_i \geq b_j - a_j = |desc(i+1)| - 2$. ◀

Later, in Claim 4, we will bound the expectation of δ_i^c, given X_i and Y_i. To do so, we will use the following simple observation.

▶ **Claim 3.** *For any $i \in \{c-1, \dots, n-1\}$, $\delta_i^c \leq \max_{j \in \{0,\dots,i-c+1\}} (x_i^{j+c} - x_i^j)$.*

Proof. Let $y_i^a = \max\{y \in Y_i' \mid y < k_{i+1}\}$. Then $y_i^{a+1} = \min\{y \in Y_i' \mid y > k_{i+1}\}$, and $\delta_i^c = y_i^{a+1} - y_i^a$. Since $Y_i' \subseteq X_i \cup \{0, n+1\}$, elements y_i^a and y_i^{a+1} are both in $X_i \cup \{x_i^0, x_i^{i+1}\}$. Let $\ell \in \{0, \dots, i\}$ such that $x_i^\ell = y_i^a$, and $r \in \{\ell+1, \dots, i+1\}$ such that $x_i^r = y_i^{a+1}$.

First assume that $r - \ell > c$. Since y_i^a and y_i^{a+1} appear consecutively in the ordered list of elements in Y_i, none of $x_i^{\ell+1}, \dots, x_i^{r-1}$ is in Y_i. Recall that $X_i \setminus Y_i$ is the set of elements in the buffer at the end of round i. Hence, all $r - \ell - 1 \geq c$ elements $x_i^{\ell+1}, \dots, x_i^{r-1}$ are in the buffer at the end of round i. But at the end of a round, the buffer can contain at most $c-1$ elements, and we have a contradiction.

Therefore, $r - \ell \leq c$. Then there exists an index $j \in \{0, \dots, \ell\}$ with $j + c \in \{r, \dots, i+1\}$. In this case $x_i^j \leq x_i^\ell = y_i^a$ and $x_i^{j+c} \geq x_i^r = y_i^{a+1}$, and thus $x_i^{j+c} - x_i^j \geq y_i^{a+1} - y_i^a = \delta_i^c$. This proves the claim. ◀

For a set $S = \{s_1, \dots, s_m\} \subseteq \{1, \dots, n\}$, where $s_1 < \dots < s_m$, let

$$\gamma(S) = \sum_{i=0}^{m} (s_{i+1} - s_i)^2, \text{ where } s_0 = 0 \text{ and } s_{m+1} = n+1.$$

For any $i \in \{0, \dots, n-1\}$ we will now bound the expected value of δ_i^c as a function of $\gamma(Y_i)$, given the sets X_i and Y_i. That bound is tight for the case $c = 1$.

▶ **Claim 4.** *For any $i \in \{c-1, \dots, n-1\}$,*

(a) $\mathbf{E}[\delta_i^c \mid X_i, Y_i] \leq \dfrac{\gamma(Y_i) - n - 1}{n - i}$;

(b) $\mathbf{E}[\delta_i^1 \mid X_i] = \dfrac{\gamma(X_i) - n - 1}{n - i}$.

Proof. Let $z = i - c + 1 = |Y_i|$, let the set of ordered elements in Y_i be $y_i^1, \ldots, y_i^z$, and let $y_i^0 = 0$ and $y_i^{z+1} = n + 1$. Therefore,

$$\sum_{j=0}^{z} (y_i^{j+1} - y_i^j) = y_i^{z+1} - y_i^0 = n + 1. \tag{2}$$

Let $\sigma(i)$ be the unique index in $\{0, \ldots, z\}$ such that $y_i^{\sigma(i)} < k_{i+1} < y_i^{\sigma(i)+1}$. Then $y_i^{\sigma(i)} = \max\{y \in \{y_i^0, \ldots, y_i^{z+1}\} \mid y < k_{i+1}\}$ and $y_i^{\sigma(i)+1} = \min\{y \in \{y_i^0, \ldots, y_i^{z+1}\} \mid y > k_{i+1}\}$. Hence,

$$\delta_i^c = y_i^{\sigma(i)+1} - y_i^{\sigma(i)}.$$

Recall that X_i is the set of keys arrived by the end of round i. Hence, given X_i, the element k_{i+1}, which arrives in round $i + 1$, is uniformly distributed over $\{1, \ldots, n\} \setminus X_i$ (the tag $(*)$ below indicates where this fact is being used). Therefore,

$$\begin{aligned}
\mathbf{E}[\delta_i^c \mid X_i, Y_i] &= \sum_{j=0}^{z} \Pr(\sigma(i) = j) \mid X_i, Y_i) \cdot (y_i^{j+1} - y_i^j) \\
&= \sum_{j=0}^{z} \Pr\left(k_{i+1} \in \{y_i^j + 1, \ldots, y_i^{j+1} - 1\} \,\middle|\, X_i, Y_i\right) \cdot (y_i^{j+1} - y_i^j) \\
&\overset{(*)}{=} \sum_{j=0}^{z} \frac{|\{y_i^j + 1, \ldots, y_i^{j+1} - 1\} \setminus X_i|}{|\{1, \ldots, n\} \setminus X_i|} \cdot (y_i^{j+1} - y_i^j) \\
&\leq \sum_{j=0}^{z} \frac{y_i^{j+1} - y_i^j - 1}{|\{1, \ldots, n\} \setminus X_i|} \cdot (y_i^{j+1} - y_i^j) \qquad (3) \\
&= \sum_{j=0}^{z} \frac{(y_i^{j+1} - y_i^j)^2 - y_i^{j+1} + y_i^j}{n - i} \\
&\overset{(2)}{=} \sum_{j=0}^{z} \frac{(y_i^{j+1} - y_i^j)^2}{n - 1} - \frac{n + 1}{n - i} \\
&= \frac{\gamma(Y_i) - n - 1}{n - i}. \qquad (4)
\end{aligned}$$

This proves Part (a) of the claim.

For Part (b) note that if the buffer has size $c = 1$, then $X_i = Y_i$. Hence, $\{y_i^j + 1, \ldots, y_i^{j+1} - 1\}$ contains no element in X_i, and so inequality (3) above holds as an equality. Now Part (b) follows from substituting x_i^j with y_i^j and X_i with Y_i in the above bound. ◀

According to Claim 2, we have

$$\Delta(n, c) - \Delta(n, 1) \leq \frac{1}{n} \sum_{i=0}^{n-1} (\mathbf{E}[\delta_i^c - \delta_i^1]).$$

In the following we will bound the expectation of the maximum of $\gamma(Y_i) - \gamma(X_i)$ for all sets $Y_i \subseteq X_i$ of size $i - c + 1$, for $i \in \{c - 1, \ldots, n - 1\}$. This implies an upper bound on $\mathbf{E}[\gamma(Y_i) - \gamma(X_i)]$, and thus by Claim 4 on $\mathbf{E}[\delta_i^c - \delta_i^1]$.

▶ **Lemma 5.** *For any $i \in \{c - 1, \ldots, n - 1\}$,*

$$\mathbf{E}[\delta_i^c - \delta_i^1] = O\left(\frac{n^2 c^2 + n^2 c \log^2 i}{i^2 (n - i)}\right).$$

First we show a high probability upper bound on the differences $x_i^{j+d} - x_i^j$, for all values of $j \in \{0, \ldots, i\}$ and $d \in \{1, \ldots, i-j+1\}$ in Claim 6 below. This allows us to bound for a random set $S \subseteq \{1, \ldots, n\}$ of m elements, the expectation of $\big(\gamma(S \setminus C) - \gamma(S)\big)$ for the "worst" subset C of S of a given size. This bound is stated in Lemma 7 below. Using this bound, we will then prove Lemma 5.

▶ **Claim 6.** *Let S be a set of $m \in \{1, \ldots, n\}$ elements chosen at random from $\{1, \ldots, n\}$ without replacement. Let $s_1, \ldots, s_m$ be the elements of S in sorted order, and let $s_0 = 0$ and $s_{m+1} = n+1$. Then for any $d \in \{1, \ldots, m+1\}$ and $j \in \{0, \ldots, m-d+1\}$,*

$$\Pr\left(s_{j+d} - s_j > \frac{8n(d+\ln m)}{m}\right) < e^{-3d} \cdot m^{-3}.$$

Proof. Let

$$b = 8n(d+\ln m)/m.$$

The distribution of the difference $s_{j+d} - s_j$ does not depend on j (see Claim 9 in the appendix), so we can assume $j = 0$. Thus, it suffices to show

$$\Pr(s_d > b) < e^{-3d} \cdot m^{-3}. \tag{5}$$

If $b > n$, then the claim is trivially true, so assume $b \le n$. Let R_t, $t \in \{1, \ldots, m\}$, be an indicator random variable, where $R_t = 1$ if and only if the t-th element chosen for S has a value of at most b. Further, let $R = R_1 + \cdots + R_m$. Then $\Pr(R_t = 1) = b/n$, and so

$$\mathbf{E}[R] = \sum_{t=1}^{m} \mathbf{E}[R_j] = m \cdot \frac{b}{n} = 8(d+\ln m).$$

Recall that $s_0, s_1, \ldots, s_{m+1}$ is an increasing sequence. Therefore, R is the smallest index in $\{1, \ldots, m\}$ such that $s_{R+1} > b$. Hence, $s_d > b$ is equivalent to $R < d$, so according to (5) it suffices to show that

$$\Pr(R < d) < e^{-3d} \cdot m^{-3}. \tag{6}$$

The random variables R_t are negatively associated [5, Section 3.1(c)], so we can use Chernoff Bounds [4, Theorem 3.1] to obtain

$$\begin{aligned}\Pr(R < d) = \Pr\left(R < \frac{E[R]}{8} - \ln m\right) &\le \Pr\big(R < E[R](1-7/8)\big) < e^{-(7/8)^2 \cdot E[R]/2} \\ &\le e^{-(7/8)^2 \cdot 8(d+\ln m)/2} < e^{-3(d+\ln m)} = e^{-3d} \cdot m^{-3}.\end{aligned}$$

This proves (6), and thus the claim. ◀

▶ **Lemma 7.** *Let S be a set of $m \in \{1, \ldots, n\}$ elements chosen at random from $\{1, \ldots, n\}$ without replacement. Then for any $\tau \in \{1, \ldots, m\}$,*

$$\mathbf{E}\left[\max_{\substack{C \subseteq S \\ |C| = \tau}} \gamma(S \setminus C) - \gamma(S)\right] = O\left(\frac{n^2\tau^2 + n^2\tau\log^2 m}{m^2}\right).$$

Proof. Let $s_1, \ldots, s_m$ be the elements of S in sorted order, and let $s_0 = 0$ and $s_{m+1} = n+1$. Define the following event,

$$\text{Event } A\text{:} \quad \forall d \in \{1, \ldots, m+1\}, i \in \{0, \ldots, m-d+1\} : s_{i+d} - s_i \le \frac{8n(d+\ln m)}{m}.$$

Using the union bound and applying Claim 6, we obtain

$$\Pr(\neg A) = \Pr\left(\exists d \in \{1,\dots,m+1\}, i \in \{0,\dots,m-d+1\} : s_{i+d} - s_i > \frac{8n(d+\ln m)}{m}\right)$$
$$\leq \sum_{d=1}^{m+1}\sum_{i=0}^{m-d+1} \Pr\left(s_{i+d} - s_i > \frac{8n(d+\ln m)}{m}\right) \leq \sum_{d=1}^{m+1}\sum_{i=0}^{m-d+1} e^{-3d}\cdot m^{-3} = O(m^{-2}). \quad (7)$$

Now consider an arbitrary set $C = \{s_{i_1},\dots,s_{i_\tau}\} \subseteq S$. Let $I = \{i_1,\dots,i_\tau\}$, and partition I into maximal subsets $I_1,\dots,I_\ell$ of consecutive indices. Further, for $j \in \{1,\dots,\ell\}$ let $\alpha_j = \min I_j - 1$ and $d_j = |I_j| + 1$. Thus,

$$C = \bigcup_{j=1}^{\ell} \{s_{\alpha_j+1}, s_{\alpha_j+2}, \dots, s_{\alpha_j+d_j-1}\},$$

and so for $S' = S \cup \{s_0, s_{m+1}\}$,

$$S' \setminus C = \{s_0, s_1, \dots, s_{\alpha_1}, s_{\alpha_1+d_1}, s_{\alpha_1+d_1+1}, \dots, s_{\alpha_2}, s_{\alpha_2+d_2}, \dots, s_{\alpha_\ell}, s_{\alpha_\ell+d_\ell}, \dots, s_{m+1}\}.$$

Each pair (s_i, s_{i+1}) over $S' \setminus C$ contributes $(s_{i+1}-s_i)^2$ to the sum $\gamma(S\setminus C)$, and it contributes the same amount to the sum $\gamma(S)$. On the other hand, each pair $(s_{\alpha_j}, s_{\alpha_j+d_j})$ over $S' \setminus C$ contributes $(s_{\alpha_j+d_j} - s_{\alpha_j})^2$ to $\gamma(S \setminus C)$, while the corresponding contribution to $\gamma(S)$ is potentially much smaller (precisely it is $\sum_{t=1}^{d_j}(s_{\alpha_j+t} - s_{\alpha_j+t-1})^2$). Therefore, ignoring these latter contributions to $\gamma(S)$, we can upper bound the difference $\gamma(S \setminus C) - \gamma(S)$ as follows:

$$\gamma(S \setminus C) - \gamma(S) \leq \sum_{j=1}^{\ell} (s_{\alpha_j+d_j} - s_{\alpha_j})^2.$$

Now, if event A occurs, then $s_{\alpha_j+d_j} - s_{\alpha_j} \leq 8n(d_j + \ln m)/m$, and so in this case

$$\gamma(S \setminus C) - \gamma(S) \leq \sum_{j=1}^{\ell} \left(\frac{8n(d_j+\ln m)}{m}\right)^2 \leq \sum_{j=1}^{\ell} \frac{128n^2\left({d_j}^2 + \ln^2 m\right)}{m^2}, \quad (8)$$

where the last inequality was obtained by using the fact that $(a+b)^2 \leq 2(a^2+b^2)$. Now recall that $d_j = |I_j| + 1$, so $d_1 + \cdots + d_\ell - \ell = |I_1 \cup \cdots \cup I_\ell| = |C| = \tau$, and $\ell \leq |C| = \tau$ (because C contains ℓ sets of consecutive indices. Therefore,

$$d_1 + \cdots + d_\ell = \tau + \ell \leq 2\tau, \text{ and } {d_1}^2 + \cdots + {d_\ell}^2 \leq (d_1 + \cdots + d_\ell)^2 = (\tau+\ell)^2 \leq 4\tau^2. \quad (9)$$

Observe that if event A occurs, bound (8) is true for every set $C \subseteq S$ of size τ. Thus, if A occurs, then combining (8) and (9), we obtain

$$\max_{\substack{C\subseteq S\\|C|=\tau}} \gamma(S\setminus C) - \gamma(S) \leq \frac{128n^2\left(4\tau^2 + \ell\ln^2 m\right)}{m^2} \leq \frac{128n^2\left(4\tau^2 + \tau\ln^2 m\right)}{m^2}. \quad (10)$$

If event A does not occur, then we can use the trivial bound

$$\max_{\substack{C\subseteq S\\|C|=\tau}} \gamma(S\setminus C) - \gamma(S) \leq \max_{\substack{C\subseteq S\\|C|=\tau}} \gamma(S\setminus C) \leq \gamma(\emptyset) = (n+1)^2. \quad (11)$$

According to (7), event $\neg A$ occurs with probability $O(m^{-2})$. Hence, (10) and (11) imply

$$\mathbf{E}[\max_{\substack{C\subseteq S\\|C|=\tau}} \gamma(S\setminus C)-\gamma(S)] \le \frac{128n^2\left(4\tau^2+\tau\ln^2 m\right)}{m^2} + (n+1)^2\cdot O(m^{-2})$$
$$= O\left(\frac{n^2\left(\tau^2+\tau\log^2 m\right)}{m^2}\right). \qquad \blacktriangleleft$$

We can now prove Lemma 5.

Proof of Lemma 5. Let $\tau = c-1$, and recall $i \in \{c-1,\ldots,n-1\}$. According to Claim 4,

$$\mathbf{E}[\delta_i^c - \delta_i^1 \mid X_i, Y_i] \le \frac{\gamma(Y_i)-\gamma(X_i)}{n-i}.$$

Recall that X_i is a set of i elements in $\{1,\ldots,n\}$ chosen uniformly at random without replacement. Moreover, $Y_i \subseteq X_i$ is chosen by the adversary, where $|X_i \setminus Y_i| = \tau$ by (1). Hence, from Lemma 7 (substituting X_i for S and i for m) we obtain

$$\mathbf{E}[\delta_i^c - \delta_i^1] \le \frac{1}{n-i}\cdot \mathbf{E}\left[\max_{\substack{C\subseteq X_i\\|C|=\tau}} \gamma(X_i\setminus C)-\gamma(X_i)\right] = O\left(\frac{1}{n-1}\cdot\frac{n^2\tau^2+n^2\tau\log^2 i}{i^2}\right).$$

Since $\tau < c$, the claim follows. $\blacktriangleleft$

For large i, close to n, the bound in Lemma 5 is too weak. Instead we use the following result which holds for $i \ge n/2$:

▶ **Lemma 8.** *Let $c < n/2$ and $i \in \{\lceil n/2\rceil,\ldots,n-1\}$. Then $\mathbf{E}[\delta_i^c] = O(c+\log n)$.*

Proof. Define

$$\text{Event } B: \quad \forall j \in \{0,\ldots,i-c+1\}: x_i^{j+c} - x_i^j \le 16(c+\ln n).$$

If B occurs, then by Claim 3, $\delta_i^c \le 16(c+\ln n)$. On the other hand, if B does not occur, then we will use the trivial bound $\delta_i^c \le n-i < n$. We will show below that

$$\Pr(\neg B) = O(n^{-2}). \tag{12}$$

Hence, we obtain

$$\mathbf{E}[\delta_i^c] \le 16(c+\ln n) + n\cdot O(n^{-2}) = O(c+\ln n).$$

Next we prove (12).

Recall that $k_1,\ldots,k_i$ are the first i elements that arrive, i.e., they are chosen uniformly at random without replacement from $\{1,\ldots,n\}$. Moreover, $x_i^1,\ldots,x_i^i$ is the sorted order of those elements. Thus, by Claim 6, for any $j \in \{0,\ldots,i-c+1\}$,

$$\Pr\left(x_i^{j+c} - x_i^j > \frac{8n(c+\ln i)}{i}\right) < e^{-3c}\cdot i^{-3}.$$

Using the assumption of this lemma, $n/2 \le i < n$, and a union bound on j, we obtain

$$\Pr\left(\exists j \in \{0,\ldots,i-c+1\}: x_i^{j+c} - x_i^j > 16(c+\ln n)\right) < e^{-3c}\cdot(n/2)^{-3}(i-c+2) = O(n^{-2}).$$

This proves (12), and thus the lemma. $\blacktriangleleft$

We now combine Lemmas 5 and 8 to prove our main result, i.e., $\Delta(n,c) = \Delta(n,1) + O(c)$.

Proof of Theorem 1. Clearly, $\Delta(n,c) < n$ for all c. Therefore, if $c \geq n/2$, the theorem is trivially true. Hence, assume $c < n/2$, and let

$$b = n - \frac{cn}{2(c+\log n)} > n - \frac{cn}{2c} = n/2 > c.$$

According to Claim 2, we have

$$\Delta(n,c) - \Delta(n,1) \leq \frac{1}{n} \cdot \sum_{i=0}^{n-1} \mathbf{E}[\delta_i^c - \delta_i^1]. \tag{13}$$

It may be useful to recall the reason for inequality (13). As explained in the proof of Claim 2, δ_i^1 is equal to the number of descendants of node k_{i+1} in the final tree in case the buffer size is 1, and δ_i^c is an upper bound for the same if the buffer has size c. Hence, the average of all values $\delta_i^c - \delta_i^1$ upper bounds the difference in the average number of descendants of all nodes between the two final trees, which in turn is equal to the difference of the average node depth between the two final trees.

We bound the sum above separately for $0 \leq i \leq c-1$, $c \leq i \leq b$, and $b < i \leq n-1$.

From the trivial bounds $\delta_i^c \leq n+1$ and $\delta_i^1 \geq 0$, we get

$$\sum_{0 \leq i \leq c-1} \mathbf{E}[\delta_i^c - \delta_i^1] \leq \sum_{0 \leq i \leq c-1} \mathbf{E}[\delta_i^c] \leq c(n+1). \tag{14}$$

By Lemma 8, we have $\mathbf{E}[\delta_i^c] = O(c + \log n)$ for $i \in \{\lceil n/2 \rceil, \ldots, n-1\}$, thus

$$\sum_{b < i \leq n-1} \mathbf{E}[\delta_i^c - \delta_i^1] = O\big((n-b) \cdot (c+\log n)\big) = O\left(cn\right). \tag{15}$$

To bound the remaining sum of $\mathbf{E}[\delta_i^c - \delta_i^1]$, for $c-1 \leq i \leq b$, first recall the following basic facts for any real $z \geq 0$ and integers $1 \leq a \leq \ell$:

$$\sum_{i=a}^{\infty} \frac{1}{i^2} \leq \frac{1}{a} + \frac{1}{a^2}; \quad \sum_{i=a}^{\ell} \frac{1}{i} \leq \frac{1}{a} + \ln(\ell/a); \quad \sum_{i=1}^{\infty} \frac{(\log i)^2}{i^2} \text{ converges; and } \ln(1+z) \leq z. \tag{16}$$

By Lemma 5,

$$\begin{aligned}
\sum_{c\le i\le b} \mathbf{E}\left[\delta_i^c - \delta_i^1\right] &= O\left(\sum_{c\le i\le b} \frac{n^2c^2 + n^2c\log^2 i}{i^2(n-i)}\right)\\
&= O\left(\sum_{c\le i\le n/2} \frac{n^2c^2 + n^2c\log^2 i}{i^2(n-i)} + \sum_{n/2<i\le b} \frac{n^2c^2 + n^2c\log^2 i}{i^2(n-i)}\right)\\
&= O\left(\sum_{c\le i\le n/2} \frac{nc^2 + nc\log^2 i}{i^2}\right) + O\left(\sum_{n/2<i\le b} \frac{c^2 + c\log^2 n}{n-i}\right)\\
&= O\left(nc\cdot\sum_{i=c}^{\infty}\left(\frac{c}{i^2} + \frac{\log^2 i}{i^2}\right)\right) + O\left((c^2 + c\log^2 n)\sum_{n-b\le j<n/2}\frac{1}{j}\right)\\
&\overset{(16)}{=} O(nc) + O\left((c^2 + c\log^2 n)\cdot\left(\frac{1}{n-b} + \ln\frac{n/2}{n-b}\right)\right)\\
&= O(nc) + O\left((c^2 + c\log^2 n)\cdot\left(1 + \ln\frac{c+\log n}{c}\right)\right)\\
&\overset{(16)}{=} O(nc) + O\left((c^2 + c\log^2 n)\cdot\left(1 + \frac{\log n}{c}\right)\right)\\
&= O(nc) + O\left(c^2 + c\log^2 n + c\log n + \log^3 n\right)\\
&= O(nc). \qquad (17)
\end{aligned}$$

Combining (13), (14), (15), and (17), we obtain $\Delta(n,c) - \Delta(n,1) = O(c)$. This completes the proof of Theorem 1. ◀

References

1. James Aspnes and Eric Ruppert. Depth of a random binary search tree with concurrent insertions. In *Proc. 30th International Symposium on Distributed Computing (DISC)*, pages 371–384, 2016.
2. Andrew Donald Booth and Andrew J.T. Colin. On the efficiency of a new method of dictionary construction. *Information and Control*, 3(4):327–334, 1960.
3. Luc Devroye. A note on the height of binary search trees. *Journal of the ACM*, 33(3):489–498, 1986.
4. Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.
5. Kumar Joag-Dev and Frank Proschan. Negative association of random variables with applications. *The Annals of Statistics*, 11(1):286–295, 1983.
6. Hosam Mahmoud Mahmoud. *Evolution of Random Search Trees*. Wiley-Interscience, 1992.
7. Bodo Manthey and Rüdiger Reischuk. Smoothed analysis of binary search trees. *Theoretical Computer Science*, 378(3):292–315, 2007.
8. Bruce Reed. The height of a random binary search tree. *Journal of the ACM*, 50(3):306–332, 2003.
9. John Michael Robson. The height of binary search trees. *Australian Computer Journal*, 11(4):151–153, 1979.
10. Peter F. Windley. Trees, forests and rearranging. *The Computer Journal*, 3(2):84–88, 1960.

A Appendix

In the following we provide a basic claim to facilitate our analysis. We believe that this simple observation should be known, but we could not find a reference for it.

▶ **Claim 9.** *Let S be a set of $m \in \{1,\dots,n\}$ elements chosen at random from $\{1,\dots,n\}$ without replacement. Let $s_1,\dots,s_m$ be the elements of S in sorted order, and let $s_0 = 0$, $s_{m+1} = n+1$. Then for any $d \in \{1,\dots,m+1\}$ and $j \in \{0,\dots,m-d+1\}$, the distribution of $s_{j+d} - s_j$ is identical to the distribution of $s_d - s_0$.*

Proof. Fix $d \in \{1,\dots,m+1\}$. Choose a set S' of $m+1$ elements in $\{0,\dots,n\}$ at random without replacement, and let $s'_0,\dots,s'_m$ be the chosen elements in sorted order. Considering the elements as chosen from the mod$(n+1)$ ring $\{0,\dots,n\}$, it is obvious by symmetry that all random variables $Z_j = (s'_{(j+d) \bmod (m+1)} - s'_j) \bmod (n+1)$, $j \in \{0,\dots,m\}$, are identically distributed. In particular, each variable Z_j, $j \in \{0,\dots,m\}$ has the same distribution as Z_0.

Then letting $s_i = s'_i - s'_0$, we have that $s_1,\dots,s_m$ have the same distribution as if they were chosen from $\{1,\dots,n\}$ at random without replacement and then sorted. Moreover, for $j \in \{0,\dots,m-d\}$ we have $s_j \le s_{j+d} \le n$, so

$$\begin{aligned} s_{j+d} - s_j = (s_{j+d} - s_j) \bmod (n+1) &= (s'_{j+d} - s'_j) \bmod (n+1) \\ &= (s'_{(j+d) \bmod (m+1)} - s'_j) \bmod (n+1) = Z_j. \end{aligned}$$

And since $s_0 \equiv s_{m+1} \pmod{n+1}$, we obtain for $j = m-d+1$ using the same calculation as above

$$s_{j+d} - s_j = (s_0 - s_j) \bmod (n+1) = (s'_{(j+d) \bmod (m+1)} - s'_j) \bmod (n+1) = Z_j.$$

Since all random variables $Z_j = s_{j+d} - s_j$, for $j \in \{0,\dots,m-d+1\}$, have the same distribution as Z_0, the claim follows. ◀

String Periods in the Order-Preserving Model

Garance Gourdel
ENS Paris-Saclay, Cachan, France
garance.gourdel@ens-paris-saclay.fr

Tomasz Kociumaka
University of Warsaw, Warsaw, Poland
kociumaka@mimuw.edu.pl

Jakub Radoszewski
University of Warsaw, Warsaw, Poland
jrad@mimuw.edu.pl

Wojciech Rytter
University of Warsaw, Warsaw, Poland
rytter@mimuw.edu.pl

Arseny Shur
Ural Federal University, Ekaterinburg, Russia
arseny.shur@urfu.ru

Tomasz Waleń
University of Warsaw, Warsaw, Poland
walen@mimuw.edu.pl

Abstract

The order-preserving model (op-model, in short) was introduced quite recently but has already attracted significant attention because of its applications in data analysis. We introduce several types of periods in this setting (op-periods). Then we give algorithms to compute these periods in time $O(n)$, $O(n \log\log n)$, $O(n \log^2 \log n / \log\log\log n)$, $O(n \log n)$ depending on the type of periodicity. In the most general variant the number of different periods can be as big as $\Omega(n^2)$, and a compact representation is needed. Our algorithms require novel combinatorial insight into the properties of such periods.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases Order-preserving Pattern Matching, Period, Efficient Algorithm

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.38

Related Version A full version of the paper is available at `https://arxiv.org/abs/1801.01404`, [29].

Funding A part of this work was done during Garance Gourdel's internship at University of Warsaw, Poland. Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter and Tomasz Waleń were supported by the Polish National Science Center, grant no. 2014/13/B/ST6/00770.

Acknowledgements A part of this work was done during the workshop "StringMasters in Warsaw 2017" that was sponsored by the Warsaw Center of Mathematics and Computer Science. The authors thank the participants of the workshop, especially Hideo Bannai and Shunsuke Inenaga, for helpful discussions.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 38; pp. 38:1–38:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Study of strings in the *order-preserving* model (*op-model*, in short) is a part of the so-called non-standard stringology. It is focused on pattern matching and repetition discovery problems in the shapes of number sequences. Here the shape of a sequence is given by the relative order of its elements. The applications of the op-model include finding trends in time series which appear naturally when considering e.g. the stock market or melody matching of two musical scores; see [32]. In such problems periodicity plays a crucial role.

One of motivations is given by the following scenario. Consider a sequence D of numbers that models a time series which is known to repeat the same shape every fixed period of time. For example, this could be certain stock market data or statistics data from a social network that is strongly dependent on the day of the week, i.e., repeats the same shape every consecutive week. Our goal is, given a fragment S of the sequence D, to discover such repeating shapes, called here *op-periods*, in S. We also consider some special cases of this setting. If the beginning of the sequence S is synchronized with the beginning of the repeating shape in D, we refer to the repeating shape as to an *initial* op-period. If the synchronization takes place also at the end of the sequence, we call the shape a *full* op-period. Finally, we also consider *sliding* op-periods that describe the case when every factor of the sequence D repeats the same shape every fixed period of time.

Order-preserving model. Let $[\![a..b]\!]$ denote the set $\{a, \ldots, b\}$. We say that two strings $X = X[1]\ldots X[n]$ and $Y = Y[1]\ldots Y[n]$ over an integer alphabet are *order-equivalent* (*equivalent* in short), written $X \approx Y$, iff $\forall_{i,j\in[\![1..n]\!]}\ X[i] < X[j] \Leftrightarrow Y[i] < Y[j]$.

▶ **Example 1.** $5\,2\,7\,5\,1\,3\,10\,3\,5 \approx 6\,4\,7\,6\,3\,5\,9\,5\,6$.

Order-equivalence is a special case of a substring consistent equivalence relation (SCER) that was defined in [37].

For a string S of length n, we can create a new string X of length n such that $X[i]$ is equal to the number of distinct symbols in S that are not greater than $S[i]$. The string X is called the *shape* of S and is denoted by $\mathit{shape}(S)$. It is easy to observe that two strings S, T are order-equivalent if and only if they have the same shape.

▶ **Example 2.** $\mathit{shape}(5\,2\,7\,5\,1\,3\,10\,3\,5) = \mathit{shape}(6\,4\,7\,6\,3\,5\,9\,5\,6) = 4\,2\,5\,4\,1\,3\,6\,3\,4$.

Periods in the op-model. We consider several notions of periodicity in the op-model, illustrated by Fig. 1. We say that a string S has a (general) *op-period* p with *shift* $s \in [\![0..p-1]\!]$ if and only if $p < |S|$ and S is a factor of a string $V_1V_2\cdots V_k$ such that:

$$|V_1| = \cdots = |V_k| = p, \quad V_1 \approx \cdots \approx V_k, \quad \text{and } S[s+1..|S|] \text{ is a prefix of } V_2\cdots V_k.$$

The *shape* of the op-period is $\mathit{shape}(V_1)$. One op-period p can have several shifts; to avoid ambiguity, we sometimes denote the op-period as (p, s). We define Shifts_p as the set of all shifts of the op-period p.

An op-period p is called *initial* if $0 \in \mathit{Shifts}_p$, *full* if it is initial and p divides $|S|$, and *sliding* if $\mathit{Shifts}_p = [\![0..p-1]\!]$. Initial and sliding op-periods are particular cases of block-based and sliding-window-based periods for SCER, both of which were introduced in [37].

Models of periodicity. In the standard model, a string S of length n has a period p iff $S[i] = S[i+p]$ for all $i = 1, \ldots, n-p$. The famous periodicity lemma of Fine and Wilf [26] states that a "long enough" string with periods p and q has also the period $\gcd(p, q)$. The

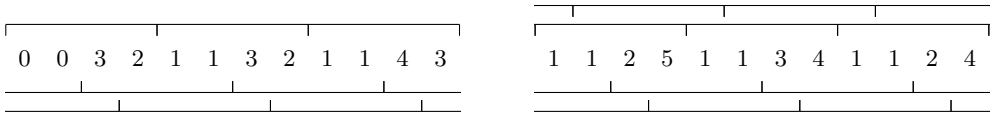


Figure 1 The string to the left has op-period 4 with three shifts: $Shifts_4 = [\![0..0]\!] \cup [\![2..3]\!]$. Due to the shift 0, the string has an initial—therefore, a full—op-period 4. The string to the right has op-period 4 with all four shifts: $Shifts_4 = [\![0..3]\!]$. In particular, 4 is a sliding op-period of the string. Notice that both strings (of length $n = 12$) have (general, sliding) periods 4, but none of them has the order-border (in the sense of [36]) of length $n - 4$.

exact bound of being "long enough" is $p + q - \gcd(p, q)$. This result was generalized to arbitrary number of periods [9, 31, 40].

Periods were also considered in a number of non-standard models. Partial words, which are strings with don't care symbols, possess quite interesting Fine–Wilf type properties, including probabilistic ones; see [4, 5, 6, 38, 39, 30]. In Section 2, we make use of periodicity graphs introduced in [38, 39]. In the abelian (jumbled) model, a version of the periodicity lemma was shown in [15] and extended in [7]. Also, algorithms for computing three types of periods analogous to full, initial, and general op-periods were designed [19, 24, 25, 33, 34, 35]. In the computation of full and initial op-periods we use some number-theoretic tools initially developed in [33, 34]. Remarkably, the fastest known algorithm for computing general periods in the abelian model has essentially quadratic time complexity [19, 35], whereas for the general op-periods we design a much more efficient solution. A version of the periodicity lemma for the parameterized model was proposed in [2].

Op-periods were first considered in [37] where initial and sliding op-periods were introduced and direct generalizations of the Fine–Wilf property to these kinds of op-periods were developed. A few distinctions between the op-periods and periods in other models should be mentioned. First, "to have a period 1" becomes a trivial property in the op-model. Second, all standard periods of a string have the "sliding" property; the first string in Fig. 1 demonstrates that this is not true for op-periods. The last distinction concerns borders. A standard period p in a string S of length n corresponds to a *border* of S of length $n - p$, which is both a prefix and a suffix of S. In the order-preserving setting, an analogue of a border is an *op-border*, that is, a prefix that is equivalent to the suffix of the same length. Op-borders have properties similar to standard borders and can be computed in $O(n)$ time [36]. However, it is no longer the case that a (general, initial, full, or sliding) op-period must correspond to an op-border; see [37].

Previous algorithmic study of the op-model. The notion of order-equivalence was introduced in [32, 36]. (However, note the related combinatorial studies, originated in [22], on containment/avoidance of shapes in permutations.) Both [32, 36] studied pattern matching in the op-model (op-pattern matching) that consists in identifying all consecutive factors of a text that are order-equivalent to a given pattern. We assume that the alphabet is integer and, as usual, that it is polynomially bounded with respect to the length of the string, which means that a string can be sorted in linear time (cf. [16]). Under this assumption, for a text of length n and a pattern of length m, [32] solve the op-pattern matching problem in $O(n + m \log m)$ time and [36] solve it in $O(n + m)$ time. Other op-pattern matching algorithms were presented in [3, 14].

An index for op-pattern matching based on the suffix tree was developed in [18]. For a text of length n it uses $O(n)$ space and answers op-pattern matching queries for a pattern of

length m in optimal, $O(m)$ time (or $O(m + Occ)$ time if we are to report all Occ occurrences). The index can be constructed in $O(n \log\log n)$ expected time or $O(n \log^2 \log n / \log\log\log n)$ worst-case time. We use the index itself and some of its applications from [18].

Other developments in this area include a multiple-pattern matching algorithm for the op-model [32], an approximate version of op-pattern matching [28], compressed index constructions [12, 21], a small-space index for op-pattern matching that supports only short queries [27], and a number of practical approaches [8, 10, 11, 13, 23].

Our results. We give algorithms to compute:

- all full op-periods in $O(n)$ time;
- the smallest non-trivial initial op-period in $O(n)$ time;
- all initial op-periods in $O(n \log\log n)$ time;
- all sliding op-periods in $O(n \log\log n)$ expected time or $O(n \log^2 \log n / \log\log\log n)$ worst-case time (and linear space);
- all general op-periods with all their shifts (compactly represented) in $O(n \log n)$ time and space. The output is the family of sets $Shifts_p$ represented as unions of disjoint intervals. The total number of intervals, over all p, is $O(n \log n)$.

In the combinatorial part, we characterize the Fine–Wilf periodicity property (aka interaction property) in the op-model in the case of coprime periods. This result is at the core of the linear-time algorithm for the smallest initial op-period.

Structure of the paper. Combinatorial foundations of our study are given in Section 2. Then in Section 3 we recall known algorithms and data structures for the op-model and develop further algorithmic tools. The remaining sections are devoted to computation of the respective types of op-periods: full and initial op-periods in Section 4, the smallest non-trivial initial op-period in Section 5, all (general) op-periods in Section 6, and sliding op-periods in Section 7. Some proofs have been omitted due to space constraints; they can be found in the preprint [29].

2 Fine–Wilf Property for Op-Periods

The following result was shown as Theorem 2 in [37]. Note that if p and q are coprime, then the conclusion is void, as every string has the op-period 1.

▶ **Theorem 3** ([37]). *Let $p > q > 1$ and $d = \gcd(p, q)$. If a string S of length $n \geq p + q - d$ has initial op-periods p and q, it has initial op-period d. Moreover, if S has length $n \geq p+q-1$ and sliding op-periods p and q, it has sliding op-period d.*

The aim of this section is to show a periodicity lemma in the case that $\gcd(p, q) = 1$.

2.1 Preliminary Notation

For a string S of length n, by $S[i]$ (for $1 \leq i \leq n$) we denote the ith letter of S and by $S[i..j]$ we denote a *factor* of S equal to $S[i] \dots S[j]$. If $i > j$, $S[i..j]$ denotes the empty string ε.

A string which is strictly increasing, strictly decreasing, or constant, is called *strictly monotone*. A *strictly monotone op-period* of S is an op-period with a strictly monotone shape. Such an op-period is called increasing (decreasing, constant) if so is its shape. Clearly, any divisor of a strictly monotone op-period is a strictly monotone op-period as well. A string S is *2-monotone* if $S = S_1 S_2$, where S_1, S_2 are strictly monotone in the same direction.

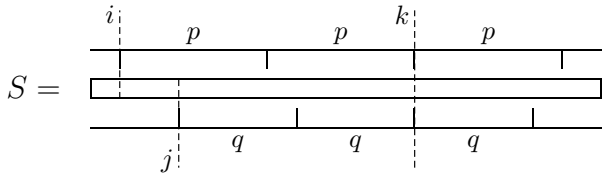


Figure 2 Op-periods (p, i) and (q, j) synchronized at position k.

Below we assume that $n > p > q > 1$. Let a string $S = S[1..n]$ have op-periods (p, i) and (q, j). If there exists a number $k \in [\![1..n-1]\!]$ such that $k \bmod p = i$ and $k \bmod q = j$, we say that these op-periods are *synchronized* and k is a *synchronization point* (see Fig. 2).

▶ Remark. The proof of Theorem 3 can be easily adapted to prove the following.

▶ **Theorem 4.** *Let $p > q > 1$ and $d = \gcd(p, q)$. If op-periods p and q of a string S of length $n \geq p + q - 1$ are synchronized, then S has op-period d, synchronized with them.*

2.2 Periodicity Theorem For Coprime Periods

For a string S, by $trace(S)$ we denote a string X of length $|S| - 1$ over the alphabet $\{+, 0, -\}$ such that:

$$X[i] = \begin{cases} + & \text{if } S[i] < S[i+1] \\ 0 & \text{if } S[i] = S[i+1] \\ - & \text{if } S[i] > S[i+1]. \end{cases}$$

▶ **Observation 5.**

(1) *A string is strictly monotone iff its trace is a unary string.*

(2) *If S has an op-period p with shift i, then $trace(S)$ "almost" has a period p, namely, $trace(S)[j] = trace(S)[k]$ for any $j, k \in [\![1..n-1]\!]$ such that $j = k \pmod p$ and $j \neq i \pmod p$. (This is because both $trace(S)[j]$ and $trace(S)[k]$ equal the sign of the difference between the same positions of the shape of the op-period of S.)*

▶ **Example 6.** Consider the string 7 5 8 1 4 6 2 4 5. It has an op-period $(3, 1)$ with shape 2 3 1. The trace of this string is:

− + − + + − + +

The positions giving the remainder 1 modulo 3 are shown in gray; the sequence of the remaining positions is periodic.

It turns out that the existence of two coprime op-periods makes a string "almost" strictly monotone. One can use periodicity graphs [38, 39] to show the following result.

▶ **Theorem 7.** *Let S be a string of length n that has coprime op-periods p and q with shifts i and j, respectively, such that $n > p > q > 1$. Then:*

(a) *if $n > pq$, then S has a strictly monotone op-period pq;*

(b) *if $2p < n \leq pq$ and the op-periods are synchronized, then S is 2-monotone;*

(c) *if $p+q < n \leq 2p$ and the op-periods are synchronized, then (q, j) is a strictly monotone op-period of S;*

(d) *if $n > \max\{2p, p+2q\}$ and the op-periods are not synchronized, then S is strictly monotone;*

(e) *if $n > 2p$, the op-periods are not synchronized, and p is initial, then S is strictly monotone;*

(f) *if $p+q < n \leq 2p$ and p is initial, then (q, j) is a strictly monotone op-period of S.*

3 Algorithmic Toolbox for Op-Model

For a string S of length n, we introduce a table op-PREF$[1..n]$ such that op-PREF$[i]$ is the length of the longest prefix of $S[i..n]$ that is equivalent to a prefix of S. It is a direct analogue of the PREF array used in standard string matching (see [20]) and can be computed similarly in $O(n)$ time using one of the standard encodings for the op-model that were used in [14, 18, 36].

▶ **Lemma 8.** *For a string of length n, the* op-PREF *table can be computed in $O(n)$ time.*

Let us mention an application of the op-PREF table that is used further in the algorithms. We denote by op-LPP$_p(S)$ ("longest op-periodic prefix") the length of the longest prefix of a string S having p as an initial op-period.

▶ **Lemma 9.** *For a string S of length n,* op-LPP$_p(S)$ *for a given p can be computed in $O(\text{op-LPP}_p(S)/p+1)$ time after $O(n)$-time preprocessing.*

Proof. We start by computing the op-PREF table for S in $O(n)$ time. We assume that op-PREF$[n+1]=0$. To compute op-LPP$_p(S)$, we iterate over positions $i = p+1, 2p+1, \ldots$ and for each of them check if op-PREF$[i] \geq p$. If i_0 is the first position for which this condition is not satisfied (possibly because $i_0 > n-p+1$), we have op-LPP$_p(S) = i_0 + \text{op-PREF}[i_0] - 1$. Clearly, this procedure works in the desired time complexity. ◀

For a string S, we define a *longest common extension* query op-LCP(i,j) in the order-preserving model as the maximum $k \geq 0$ such that $S[i..i+k-1] \approx S[j..j+k-1]$. Symmetrically, op-LCS$(i,j)$ is the maximum $k \geq 0$ such that $S[i-k+1..i] \approx S[j-k+1..j]$.

Similarly as in the standard model [17], LCP-queries in the op-model can be answered using lowest common ancestor (LCA) queries in the op-suffix tree; see the following lemma.

▶ **Lemma 10.** *For a string of length n, after preprocessing in $O(n \log\log n)$ expected time or in $O(n \log^2 \log n / \log\log\log n)$ worst-case time one can answer* op-LCP*-queries in $O(1)$ time.*

The factor $S[i..i+2p-1]$ is called an order-preserving square (*op-square*) iff $S[i..i+p-1] \approx S[i+p..i+2p-1]$. For a string S of length n, we define the set

$$\textit{op-Squares}_p = \{i \in [\![1..n-2p+1]\!] : S[i..i+2p-1] \text{ is an op-square}\}.$$

Op-squares were first defined in [18] where an algorithm computing all the sets $\textit{op-Squares}_p$ for a string of length n in $O(n \log n + \sum_p |\textit{op-Squares}_p|)$ time was shown.

We say that an op-square $S[i..i+2p-1]$ is *right shiftable* if $S[i+1..i+2p]$ is an op-square and *right non-shiftable* otherwise. Similarly, we say that the op-square is *left shiftable* if $S[i-1..i+2p-2]$ is an op-square and *left non-shiftable* otherwise. Using the approach of [18], one can show the following lemma.

▶ **Lemma 11.** *All the (left and right) non-shiftable op-squares in a string of length n can be computed in $O(n \log n)$ time.*

4 Computing All Full and Initial Op-Periods

For a string S of length n, we define op-PREF$'[i]$ for $i = 0, \ldots, n$ as:

$$\text{op-PREF}'[i] = \begin{cases} n & \text{if op-PREF}[i+1] = n-i \\ \text{op-PREF}[i+1] & \text{otherwise.} \end{cases}$$

Algorithm 1: Computing All Initial Op-Periods of S.

```
1 T := op-PREF';
2 for j := n down to 2 do
3     foreach prime divisor q of j do
4         P[j/q] := min(P[j/q], P[j]);
5 for p := 1 to n do
6     if P[p] ≥ p then p is an initial op-period;
```

Here we assume that op-PREF$[n+1] = 0$. In the computation of full and initial op-periods we heavily rely on this table according to the following obvious observation.

▶ **Observation 12.** *p is an initial op-period of a string S of length n if and only if* op-PREF$'[ip] \geq p$ *for all* $i = 1, \ldots, \lfloor n/p \rfloor$.

4.1 Computing Initial Op-Periods

Let us introduce an auxiliary array $P[0..n]$ such that:

$$P[p] = \min\{\text{op-PREF}'[ip] \,:\, i = 1, \ldots, \lfloor n/p \rfloor\}.$$

Straight from Observation 12 we have:

▶ **Observation 13.** *p is an initial period of S if and only if $P[p] \geq p$.*

The table T could be computed straight from definition in $O(n \log n)$ time. We improve this complexity to $O(n \log\log n)$ by employing Eratosthenes's sieve. The sieve computes, in particular, for each $j = 1, \ldots, n$ a list of all distinct prime divisors of j. We use these divisors to compute the table via dynamic programming in a right-to-left scan, as shown in Algorithm 1.

▶ **Theorem 14.** *All initial op-periods of a string of length n can be computed in $O(n \log\log n)$ time.*

Proof. By Lemma 8, the op-PREF table for the string—hence, the op-PREF$'$ table—can be computed in $O(n)$ time. Then we use Algorithm 1. Each prime number $q \leq n$ has at most $\frac{n}{q}$ multiples below n. Therefore, the complexity of Eratosthenes's sieve and the number of updates on the table T in the algorithm is $\sum_{q \in Primes, q \leq n} \frac{n}{q} = O(n \log\log n)$; see [1]. ◀

4.2 Computing Full Op-Periods

Let us recall the following auxiliary data structure for efficient gcd-computations that was developed in [34]. We will only need a special case of this data structure to answer queries for $\gcd(x, n)$.

▶ **Fact 15** (Theorem 4 in [34]). *After $O(n)$-time preprocessing, given any $x, y \in \{1, \ldots, n\}$, the value $\gcd(x, y)$ can be computed in constant time.*

Let $Div(i)$ denote the set of all positive divisors of i. In the case of full op-periods we only need to compute $P[p]$ for $p \in Div(n)$. As in Algorithm 1, we start with $T =$ op-PREF$'$. Then we perform a preprocessing phase that shifts the information stored in the array from

Algorithm 2: Computing All Full Op-Periods of S.

```
1 T := op-PREF';
2 for i := 1 to n do
3     k := gcd(i, n);
4     P[k] := min(P[k], P[i]);
5 foreach i ∈ Div(n) in decreasing order do
6     foreach d ∈ Div(i) do
7         P[d] := min(P[d], P[i]);
8 foreach p ∈ Div(n) do
9     if P[p] ≥ p then p is a full op-period;
```

indices $i \notin Div(n)$ to indices $\gcd(i, n) \in Div(n)$. It is based on the fact that for $d \in Div(n)$, $d \mid i$ if and only if $d \mid \gcd(i, n)$. Finally, we perform right-to-left processing as in Algorithm 1. However, this time we can afford to iterate over all divisors of elements from $Div(n)$. Thus we arrive at the pseudocode of Algorithm 2.

▶ **Theorem 16.** *All full op-periods of a string of length n can be computed in $O(n)$ time.*

Proof. We apply Algorithm 2. The complexity of the first for-loop is $O(n)$ by Fact 15. The second for-loop works in $O(n)$ time as the sizes of the sets $Div(n)$, $Div(i)$ are $O(\sqrt{n})$ and the elements of these sets can be enumerated in $O(\sqrt{n})$ time as well. ◀

5 Computing Smallest Non-Trivial Initial Op-Period

If a string is not strictly monotone itself, it has $O(n)$ such op-periods and they can all be computed in $O(n)$ time. We use this as an auxiliary routine in the computation of the smallest initial op-period that is greater than 1.

▶ **Theorem 17.** *If a string of length n is not strictly monotone, all of its strictly monotone op-periods can be computed in $O(n)$ time.*

Let us start with the following simple property.

▶ **Lemma 18.** *The shape of the smallest non-trivial initial op-period of a string has no shorter non-trivial full op-period.*

Proof. A full op-period of the initial op-period of a string S is an initial op-period of S. ◀

Now we can state a property of initial op-periods, implied by Theorem 7, that is the basis of the algorithm.

▶ **Lemma 19.** *If a string of length n has initial op-periods $p > q > 1$ such that $p + q < n$ and $\gcd(p, q) = 1$, then q is strictly monotone.*

Proof. Let us consider three cases. If $n > pq$, then by Theorem 7(a), both p and q are strictly monotone. If $2p < n \leq pq$, then Theorem 7(e) implies that $S[1..pq - 1]$ is strictly monotone, hence p and q are strictly monotone as well. Finally, if $p + q < n \leq 2p$, we have that q is strictly monotone by Theorem 7(f). ◀

Algorithm 3: Computing the Smallest Non-Trivial Initial Op-Period of S.

```
1 if S has a non-trivial strictly monotone op-period then
2     return smallest such op-period;                    ▷ Theorem 17
3 p := the length of the longest monotone prefix of S plus 1;
4 while p ≤ n do
5     k := op-LPP_p(S);
6     if k = n then return p;
7     p := max(p + 1, k − p − 1);
8 return min(p_mon, n);
```

▶ **Theorem 20.** *The smallest initial op-period $p > 1$ of a string S of length n can be computed in $O(n)$ time.*

Proof. We follow the lines of Algorithm 3. If S is not strictly monotone itself, we can compute the smallest non-trivial strictly monotone initial op-period of S using Theorem 17. Otherwise, the smallest such op-period is 2. If S has a non-trivial strictly monotone initial op-period and the smallest such op-period is $q > 1$, then none of $2, \ldots, q-1$ is an initial op-period of S. Hence, we can safely return q.

Let us now focus on the correctness of the while-loop. The invariant is that there is no initial op-period of S that is smaller than p. If the value of $k = \text{op-LPP}_p(S)$ equals n, then p is an initial op-period of S and we can safely return it. Otherwise, we can advance p by 1. There is also no smallest initial op-period p' such that $p < p' < k - p - 1$. Indeed, Lemma 19 would imply that p is strictly monotone if $\gcd(p, p') = 1$ (which is impossible due to the initial selection of p) and Theorem 3 would imply an initial op-period of $S[1..p']$ that is smaller than p' and divides p' if $\gcd(p, p') > 1$ (which is impossible due to Lemma 18). This justifies the way p is increased.

Now let us consider the time complexity of the algorithm. The algorithm for strictly monotone op-periods of Theorem 17 works in $O(n)$ time. By Lemma 9, k can be computed in $O(k/p + 1)$ time. If $k \leq 3p$, this is $O(1)$. Otherwise, p at least doubles; let p' be the new value of p. Then $O(k/p + 1) = O((p + p' - 1)/p + 1) = O(p' + 1)$. The case that p doubles can take place at most $O(\log n)$ times and the total sum of p' over such cases is $O(n)$. ◀

6 Computing All Op-Periods

An *interval representation* of a set X of integers is $X = [\![i_1..j_1]\!] \cup [\![i_2..j_2]\!] \cup \cdots \cup [\![i_k..j_k]\!]$ where $j_1 + 1 < i_2$, …, $j_{k-1} + 1 < i_k$; k is called the *size* of the representation.

Our goal is to compute a *compact representation* of all the op-periods of a string that contains, for each op-period p, an interval representation of the set Shifts_p.

For an integer set X, by $X \bmod p$ we denote the set $\{x \bmod p : x \in X\}$. The following technical lemma provides efficient operations on interval representations of sets.

▶ **Lemma 21.**

(a) *Assume that X and Y are two sets with interval representations of sizes x and y, respectively. Then the interval representation of the set $X \cap Y$ can be computed in $O(x + y)$ time.*

(b) *Assume that $X_1, \ldots, X_k \subseteq [\![0..n]\!]$ are sets with interval representations of sizes $x_1, \ldots, x_k$ and $p_1, \ldots, p_k$ be positive integers. Then the interval representations of all the sets $X_1 \bmod p_1, \ldots, X_k \bmod p_k$ can be computed in $O(x_1 + \cdots + x_k + k + n)$ time.*

Algorithm 4: Computing a Compact Representation of All Op-Periods.

```
1  Compute op-Squares_p for all p = 1, ..., n;                    ▷ Lemma 22
2  for p := 1 to n do
3      N_p := [[1..n − 2p + 1]] \ op-Squares_p;
4      k := op-LCP(1, p + 1); ℓ := op-LCS(n, n − p);
5      if k = n − p then B_p := C_p := [[1..n]];
6      else B_p := [[1..k]]; C_p := [[n − ℓ + 1..n]];
7  for p := 1 to n simultaneously do
8      N_p := {(x − 1) mod p : x ∈ N_p}; B_p := B_p mod p; C_p := C_p mod p; ▷ Lemma 21(b)
9  Shifts_1 := [[0]];
10 for p := 2 to n do
11     A_p := [[0..p − 1]] \ N_p;
12     Shifts_p := A_p ∩ B_p ∩ C_p;                                ▷ Lemma 21(a)
13 return Shifts_p for p = 1, ..., n;
```

▶ **Lemma 22.** *For a string of length n, interval representations of the sets op-Squares$_p$ for all $1 \le p \le n/2$ can be computed in $O(n \log n)$ time.*

Proof. Let us define the following two auxiliary sets.

$$\mathcal{L}_p = \{i \in [\![1..n-2p+1]\!] : S[i..i+2p-1] \text{ is a left non-shiftable op-square}\}$$
$$\mathcal{R}_p = \{i \in [\![1..n-2p+1]\!] : S[i..i+2p-1] \text{ is a right non-shiftable op-square}\}.$$

By Lemma 11, all the sets $\mathcal{L}_p$ and $\mathcal{R}_p$ can be computed in $O(n \log n)$ time. In particular, $\sum_p |\mathcal{L}_p| = O(n \log n)$.

Let us note that, for each p, $|\mathcal{L}_p| = |\mathcal{R}_p|$. Thus let $\mathcal{L}_p = \{\ell_1, \ldots, \ell_k\}$ and $\mathcal{R}_p = \{r_1, \ldots, r_k\}$. The interval representation of the set *op-Squares*$_p$ is $[\![\ell_1..r_1]\!] \cup \cdots \cup [\![\ell_k..r_k]\!]$. Clearly, it can be computed in $O(|\mathcal{L}_p|)$ time. ◀

We will use the following characterization of op-periods.

▶ **Observation 23.** *p is an op-period of S with shift i if and only if all the following conditions hold:*

(A) *$S[i+1+kp..i+(k+2)p]$ is an op-square for every $0 \le k \le (n-2p-i)/p$,*

(B) $\text{op-LCP}(1, p+1) \ge \min(i, n-p)$,

(C) $\text{op-LCS}(n, n-p) \ge \min((n-i) \bmod p, n-p)$.

▶ **Theorem 24.** *A representation of size $O(n \log n)$ of all the op-periods of a string of length n can be computed in $O(n \log n)$ time.*

Proof. We use Algorithm 4. The sets $\mathcal{A}_p$, $\mathcal{B}_p$, and $\mathcal{C}_p$ describe the sets of shifts i that satisfy conditions (A), (B), and (C) from Observation 23, respectively.

A crucial role is played by the set $\mathcal{N}_p$ of all positions which are *not* the beginnings of op-squares of length $2p$. It is computed as a complement of the set *op-Squares*$_p$.

Operations "mod" on sets are performed simultaneously using Lemma 21(b). All sets $\mathcal{A}_p$, $\mathcal{B}_p$, $\mathcal{C}_p$ have $O(n \log n)$-sized representations. This guarantees $O(n \log n)$ time. ◀

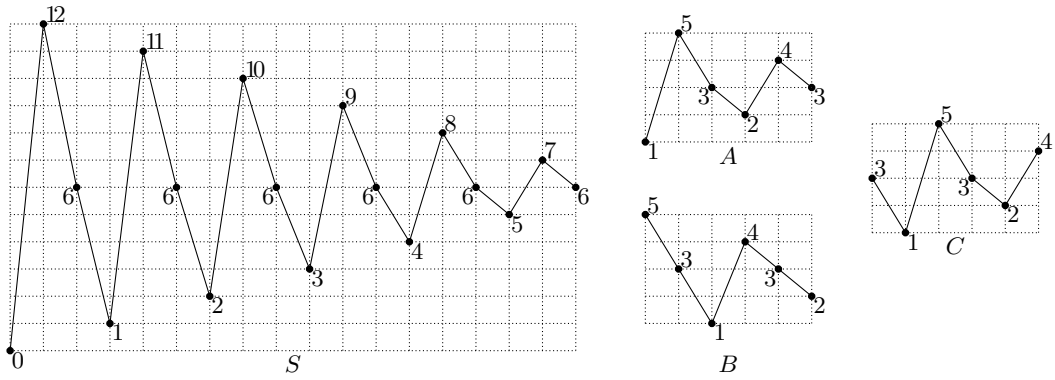


Figure 3 A string $S = 0\,12\,6\,1\,11\,6\,2\,10\,6\,3\,9\,6\,4\,8\,6\,5\,7\,6$ is graphically illustrated above (the ith point has coordinates $(i, S[i])$). We have $SH_6 = ABCABCABCA$, where $A = 1\,5\,3\,2\,4\,3$, $B = 5\,3\,1\,4\,3\,2$, and $C = 3\,1\,5\,3\,2\,4$. The shortest period of SH_6 is 3. Hence, 6 is a sliding op-period of S. Moreover, Lemma 27b implies that 3 is a period of SH_3, hence a sliding op-period of S.

7 Computing Sliding Op-Periods

For a string S of length n, we define a family of strings $SH_1, \ldots, SH_n$ such that $SH_k[i] = \mathit{shape}(S[i..i+k-1])$ for $1 \le i \le n-k+1$. Note that the characters of the strings are shapes. Moreover, the total length of strings SH_k is quadratic in n, so we will not compute those strings explicitly. Instead, we use the following observation to test if two symbols are equal.

▶ **Observation 25.** *$SH_k[i] = SH_k[i']$ if and only if* op-LCP$(i, i') \ge k$.

Sliding op-periods admit an elegant characterization based on SH_k; see Figure 3.

▶ **Lemma 26.** *An integer p, $1 \le p \le n$, is a sliding op-period of S if and only if $p \le \frac{1}{2}n$ and p is a period of SH_p, or $p > \frac{1}{2}n$ and $S[1..n-p] \approx S[p+1..n]$.*

For a string X, we denote the shortest period of X by $\mathsf{per}(X)$.

▶ **Lemma 27.** *Suppose that $p = \mathsf{per}(SH_k[1..\ell]) < \ell$. Then*
(a) *p is also a period of $SH_{k'}[1..\ell+k-k']$ for $1 \le k' \le k$,*
(b) *$q = \mathsf{per}(SH_k[1..\ell+1])$ satisfies $p = q$ or $p + q > \ell$.*

We introduce a two-dimensional table PER, where:

$$PER[k,\ell] = \mathsf{per}(SH_k[1..\ell]) \text{ if } \mathsf{per}(SH_k[1..\ell]) \le \frac{1}{3}\ell, \text{ and } PER[k,\ell] = \perp(\text{undefined}) \text{ otherwise.}$$

The size of PER is quadratic in n. However, Algorithm 5 computes PER column after column, keeping only the current column $P = PER[\cdot, \ell]$. The total number of differences between consecutive columns is linear. Hence, any requested $O(n)$ values $PER[k,\ell]$ can be computed in $O(n)$ time. We also use an analogous table PER^R for the reverse string S^R.

▶ **Lemma 28.** *Algorithm 5 is correct, that is, it satisfies the invariant.*

Proof. First, observe that the invariant is satisfied after the first iteration. This is because $\mathsf{per}(SH_k[1..1]) = 1$ for each k and the initial values are not changed during this iteration.

Thus, our task is to prove that the invariant is preserved after each subsequent ℓth iteration. Let $t = \min\{k : PER[k, \ell-1] = \perp\}$ and $t' = \min\{k : PER[k,\ell] = \perp\}$.

Algorithm 5: Computation of $PER[\cdot, \ell]$ from $PER[\cdot, \ell - 1]$.

```
1 P[1..n] := [⊥, ..., ⊥]; t := 1; ℓ' := 3;
2 for ℓ := 1 to n do
3     if t > 1 and SH_{t-1}[ℓ] ≠ SH_{t-1}[ℓ − P[t − 1]] then
4         t := t − 1; P[t] := ⊥; ℓ' := 2ℓ;
5     if ℓ ≥ ℓ' then
6         while per(SH_t[1..ℓ]) = (1/3)ℓ do
7             P[t] := (1/3)ℓ; t := t + 1; ℓ' := 2ℓ;
      ▷ Invariant: P[k] = PER[k, ℓ] , t = min{k : P[k] = ⊥}, and per(SH_t[1..ℓ]) ≥ (1/3)ℓ'.
```

Algorithm 6: Computing the sliding op-periods $p \leq \frac{1}{2}n$.

```
1 p := 1;
2 while p ≤ (1/2)n do
3     if (q := PER[p, n − 2p + 1]) = PER^R[p, n − 2p + 1] ≠ ⊥ then
4         if p is a period of SH_p[1..p + q] then report p;
5         p := min{p' > p : p' is a period of SH_p[1..p + 2q]}
6     else if PER[p, ⌈(3/4)(n − 2p + 1)⌉] = PER^R[p, ⌈(3/4)(n − 2p + 1)⌉] ≠ ⊥ then p := p + 1;
7     else
8         if p is a period of SH_p then report p;
9         p := min{p' > p : p' is a period of SH_p};
```

First, we consider the values $PER[k, \ell]$ for $k < t$. For this, we assume $t > 1$ and denote $p = PER[t - 1, \ell - 1]$. Since p is a period of $SH_{t-1}[1..\ell - 1]$, Lemma 27a yields that p is also a period of $SH_k[1..\ell]$ for $k < t - 1$. We apply Lemma 27b for $p' = \mathsf{per}(SH_k[1..\ell - 1])$. Since $p' + p \leq \ell - 1$, we conclude that $p' = \mathsf{per}(SH_k[1..\ell])$, i.e., $PER[k, \ell - 1] = p' = PER[k, \ell]$. Now, we consider the value $PER[t - 1, \ell]$. Lemma 27b, applied for $p = \mathsf{per}(SH_{t-1}[1..\ell - 1])$ and $q = \mathsf{per}(SH_{t-1}[1..\ell])$, yields $p = q$ or $p + q \geq \ell$. To verify the first case, we check whether $SH_{t-1}[\ell] = SH_{t-1}[\ell - p]$. In the second case, we conclude that $q \geq \frac{2}{3}\ell$, so $PER[t - 1, \ell] = \bot$ (and $\ell' := 2\ell$ is also set correctly).

Next, we consider the values $PER[k, \ell]$ for $k \geq t$. Since $PER[k, \ell - 1] = \bot$, we have $PER[k, \ell] = \bot$ or $PER[k, \ell] = \frac{1}{3}\ell$. More precisely, $PER[k, \ell] = \bot$ for $k \geq t'$ and $PER[k, \ell] = \frac{1}{3}\ell$ for $t \leq k < t'$. Thus, we check if $\mathsf{per}(SH_k[1..\ell]) = \frac{1}{3}\ell$ for subsequent values $k \geq t$. Since $\mathsf{per}(SH_t[1..\ell]) \geq \frac{1}{3}\ell'$, no verification is needed if $\ell < \ell'$. To complete the proof, we need to show that the update $\ell' := 2\ell$ is valid if $t' > t$. For a proof by contradiction suppose that $r := \mathsf{per}(SH_{t'}[1..\ell]) < \frac{2}{3}\ell$. By Lemma 27a, r is a period of $SH_t[1..\ell]$. Since $r + \frac{1}{3}\ell \leq \ell$, Periodicity Lemma yields $\frac{1}{3}\ell \mid r$, and thus $r = \frac{1}{3}\ell$, which contradicts the definition of t'. ◀

▶ **Lemma 29.** *Algorithm 5 can be implemented in time $O(n)$ plus the time to answer $O(n)$* op-LCP *queries in S.*

▶ **Lemma 30.** *Algorithm 6 is correct, that is, it reports all sliding op-periods $p \leq \frac{1}{2}n$ of S.*

Proof. Let p_i be the value of p at the beginning of the ith iteration of the while-loop and let $\ell_i = n - 2p_i + 1$. We shall prove that p_i is reported if and only if it is a sliding op-period and that there is no sliding op-period strictly between p_i and p_{i+1}.

First, suppose that $q = \mathsf{per}(SH_{p_i}[1..\ell_i]) = \mathsf{per}(SH_{p_i}[p_i + 1..p_i + \ell_i]) \leq \frac{1}{3}\ell_i$, i.e., we are in the first branch. If $SH_{p_i}[1..q] = SH_{p_i}[p_i + 1..p_i + q]$, then we must have $SH_{p_i}[1..\ell_i] =$

$SH_{p_i}[p_i+1..p_i+\ell_i]$, i.e., p_i is a period of $SH_{p_i} = SH_{p_i}[1..p_i+\ell_i]$ and p_i is a sliding op-period due to Lemma 26. Moreover, any sliding op-period $p' > p_i$ must be a period of SH_{p_i} (and, in particular, of $SH_{p_i}[1..p_i+2q]$) due to Lemma 27a. Consequently, $p' \geq p_{i+1}$, as claimed.

In the second branch we only need to prove that $SH_{p_i}[1..\ell_i] \neq SH_{p_i}[p_i+1..p_i+\ell_i]$. For a proof by contradiction, suppose that we have an equality. The condition from Line 6 means that the length-$\lceil \frac{3}{4}\ell_i \rceil$ prefix and suffix of $SH_{p_i}[1..\ell_i] = SH_{p_i}[p_i+1..p_i+\ell_i]$ has the common shortest period $q \leq \frac{1}{3}\lceil \frac{3}{4}\ell_i \rceil \leq \lceil \frac{1}{4}\ell_i \rceil$. The prefix and the suffix overlap by at least $\lceil \frac{1}{2}\ell_i \rceil$ characters, so we actually have $q = \mathsf{per}(SH_{p_i}[1..\ell_i]) = \mathsf{per}(SH_{p_i}[p_i+1..p_i+\ell_i])$. Hence, in that case we would be in the first branch.

Finally, in the third branch we directly use Lemma 26 to check if p_i is a sliding op-period. Moreover, if $p' > p_i$ is also a sliding op-period, then p' is a period of SH_{p_i}, i.e., $p' \geq p_{i+1}$. ◀

Let us observe that $PER[k,\ell]$ and $PER^R[k,\ell]$ is used in Algorithm 6 only for $\ell = n-2k+1$ or $\ell = \lceil \frac{3}{4}(n-2k+1) \rceil$. These $O(n)$ values can be computed in $O(n)$ time using Algorithm 5. In [29] we show the following lemma.

▶ **Lemma 31.** *Algorithm 6 can be implemented in time $O(n)$ plus the time to answer $O(n)$* op-LCP *and* op-LCS *queries in S.*

▶ **Theorem 32.** *All sliding op-periods of a string of length n can be computed in $O(n)$ space and $O(n \log\log n)$ expected time or $O(n \log^2 \log n / \log\log\log n)$ worst-case time.*

Proof. First, we apply Lemma 10 so that op-LCP and op-LCS queries can be answered in $O(1)$ time. Next, we run Algorithm 6 to report sliding op-periods $p \leq \frac{1}{2}n$. Then, we iterate over $p > \frac{1}{2}n$ and report p if op-LCP$(1, p+1) = n-p$. Correctness follows from Lemmas 30 and 26. The overall time is $O(n)$ (Lemma 31) plus the preprocessing time of Lemma 10. ◀

References

1 Tom M. Apostol. *Introduction to Analytic Number Theory.* Undergraduate Texts in Mathematics, Springer, 1976.

2 Alberto Apostolico and Raffaele Giancarlo. Periodicity and repetitions in parameterized strings. *Discrete Applied Mathematics*, 156(9):1389–1398, 2008. doi:10.1016/j.dam.2006.11.017.

3 Djamal Belazzougui, Adeline Pierrot, Mathieu Raffinot, and Stéphane Vialette. Single and multiple consecutive permutation motif search. In Leizhen Cai, Siu-Wing Cheng, and Tak Wah Lam, editors, *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, volume 8283 of *Lecture Notes in Computer Science*, pages 66–77. Springer, 2013. doi:10.1007/978-3-642-45030-3_7.

4 Jean Berstel and Luc Boasson. Partial words and a theorem of fine and wilf. *Theor. Comput. Sci.*, 218(1):135–141, 1999. doi:10.1016/S0304-3975(98)00255-2.

5 Francine Blanchet-Sadri, Deepak Bal, and Gautam Sisodia. Graph connectivity, partial words, and a theorem of fine and wilf. *Inf. Comput.*, 206(5):676–693, 2008. doi:10.1016/j.ic.2007.11.007.

6 Francine Blanchet-Sadri and Robert A. Hegstrom. Partial words and a theorem of fine and wilf revisited. *Theor. Comput. Sci.*, 270(1-2):401–419, 2002. doi:10.1016/S0304-3975(00)00407-2.

7 Francine Blanchet-Sadri, Sean Simmons, Amelia Tebbe, and Amy Veprauskas. Abelian periods, partial words, and an extension of a theorem of fine and wilf. *RAIRO - Theor. Inf. and Applic.*, 47(3):215–234, 2013. doi:10.1051/ita/2013034.

8 Domenico Cantone, Simone Faro, and M. Oguzhan Külekci. An efficient skip-search approach to the order-preserving pattern matching problem. In Jan Holub and Jan Zdárek, editors, *Proceedings of the Prague Stringology Conference 2015, Prague, Czech Republic, August 24-26, 2015*, pages 22–35. Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2015. URL: http://www.stringology.org/event/2015/p04.html.

9 Maria Gabriella Castelli, Filippo Mignosi, and Antonio Restivo. Fine and wilf's theorem for three periods and a generalization of sturmian words. *Theor. Comput. Sci.*, 218(1):83–94, 1999. doi:10.1016/S0304-3975(98)00251-5.

10 Tamanna Chhabra, Simone Faro, M. Oguzhan Külekci, and Jorma Tarhio. Engineering order-preserving pattern matching with SIMD parallelism. *Softw., Pract. Exper.*, 47(5):731–739, 2017. doi:10.1002/spe.2433.

11 Tamanna Chhabra, Emanuele Giaquinta, and Jorma Tarhio. Filtration algorithms for approximate order-preserving matching. In Costas S. Iliopoulos, Simon J. Puglisi, and Emine Yilmaz, editors, *String Processing and Information Retrieval - 22nd International Symposium, SPIRE 2015, London, UK, September 1-4, 2015, Proceedings*, volume 9309 of *Lecture Notes in Computer Science*, pages 177–187. Springer, 2015. doi:10.1007/978-3-319-23826-5_18.

12 Tamanna Chhabra, M. Oguzhan Külekci, and Jorma Tarhio. Alternative algorithms for order-preserving matching. In Jan Holub and Jan Zdárek, editors, *Proceedings of the Prague Stringology Conference 2015, Prague, Czech Republic, August 24-26, 2015*, pages 36–46. Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2015. URL: http://www.stringology.org/event/2015/p05.html.

13 Tamanna Chhabra and Jorma Tarhio. A filtration method for order-preserving matching. *Inf. Process. Lett.*, 116(2):71–74, 2016. doi:10.1016/j.ipl.2015.10.005.

14 Sukhyeun Cho, Joong Chae Na, Kunsoo Park, and Jeong Seop Sim. A fast algorithm for order-preserving pattern matching. *Inf. Process. Lett.*, 115(2):397–402, 2015. doi:10.1016/j.ipl.2014.10.018.

15 Sorin Constantinescu and Lucian Ilie. Fine and Wilf's theorem for Abelian periods. *Bulletin of the EATCS*, 89:167–170, 2006.

16 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

17 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.

18 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Alessio Langiu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Order-preserving indexing. *Theor. Comput. Sci.*, 638:122–135, 2016. doi:10.1016/j.tcs.2015.06.050.

19 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Jakub Pachocki, Jakub Radoszewski, Wojciech Rytter, Wojciech Tyczynski, and Tomasz Walen. A note on efficient computation of all abelian periods in a string. *Inf. Process. Lett.*, 113(3):74–77, 2013. doi:10.1016/j.ipl.2012.11.001.

20 Maxime Crochemore and Wojciech Rytter. *Jewels of Stringology*. World Scientific, 2003.

21 Gianni Decaroli, Travis Gagie, and Giovanni Manzini. A compact index for order-preserving pattern matching. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2017 Data Compression Conference, DCC 2017, Snowbird, UT, USA, April 4-7, 2017*, pages 72–81. IEEE, 2017. doi:10.1109/DCC.2017.35.

22 Sergi Elizalde and Marc Noy. Consecutive patterns in permutations. *Advances in Applied Mathematics*, 30(1):110–125, 2003. doi:10.1016/S0196-8858(02)00527-4.

23 Simone Faro and M. Oguzhan Külekci. Efficient algorithms for the order preserving pattern matching problem. In Riccardo Dondi, Guillaume Fertin, and Giancarlo Mauri, editors, *Algorithmic Aspects in Information and Management - 11th International Conference, AAIM 2016, Bergamo, Italy, July 18-20, 2016, Proceedings*, volume 9778 of *Lecture Notes in Computer Science*, pages 185–196. Springer, 2016. doi:10.1007/978-3-319-41168-2_16.

24 Gabriele Fici, Thierry Lecroq, Arnaud Lefebvre, and Élise Prieur-Gaston. Algorithms for computing abelian periods of words. *Discrete Applied Mathematics*, 163:287–297, 2014. doi:10.1016/j.dam.2013.08.021.

25 Gabriele Fici, Thierry Lecroq, Arnaud Lefebvre, Élise Prieur-Gaston, and William F. Smyth. A note on easy and efficient computation of full abelian periods of a word. *Discrete Applied Mathematics*, 212:88–95, 2016. doi:10.1016/j.dam.2015.09.024.

26 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proc. Amer. Math. Soc.*, 16:109–114, 1965.

27 Travis Gagie, Giovanni Manzini, and Rossano Venturini. An encoding for order-preserving matching. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 38:1–38:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.38.

28 Pawel Gawrychowski and Przemyslaw Uznanski. Order-preserving pattern matching with k mismatches. *Theor. Comput. Sci.*, 638:136–144, 2016. doi:10.1016/j.tcs.2015.08.022.

29 Garance Gourdel, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Arseny Shur, and Tomasz Waleń. String periods in the order-preserving model. ArXiv preprint. URL: https://arxiv.org/abs/1801.01404.

30 Lidia A. Idiatulina and Arseny M. Shur. Periodic partial words and random bipartite graphs. *Fundam. Inform.*, 132(1):15–31, 2014. doi:10.3233/FI-2014-1030.

31 Jacques Justin. On a paper by castelli, mignosi, restivo. *ITA*, 34(5):373–377, 2000. doi:10.1051/ita:2000122.

32 Jinil Kim, Peter Eades, Rudolf Fleischer, Seok-Hee Hong, Costas S. Iliopoulos, Kunsoo Park, Simon J. Puglisi, and Takeshi Tokuyama. Order-preserving matching. *Theor. Comput. Sci.*, 525:68–79, 2014. doi:10.1016/j.tcs.2013.10.006.

33 Tomasz Kociumaka, Jakub Radoszewski, and Wojciech Rytter. Fast algorithms for abelian periods in words and greatest common divisor queries. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, volume 20 of *LIPIcs*, pages 245–256. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.STACS.2013.245.

34 Tomasz Kociumaka, Jakub Radoszewski, and Wojciech Rytter. Fast algorithms for abelian periods in words and greatest common divisor queries. *J. Comput. Syst. Sci.*, 84:205–218, 2017. doi:10.1016/j.jcss.2016.09.003.

35 Tomasz Kociumaka, Jakub Radoszewski, and Bartlomiej Wisniewski. Subquadratic-time algorithms for abelian stringology problems. In Ilias S. Kotsireas, Siegfried M. Rump, and Chee K. Yap, editors, *Mathematical Aspects of Computer and Information Sciences - 6th International Conference, MACIS 2015, Berlin, Germany, November 11-13, 2015, Revised Selected Papers*, volume 9582 of *Lecture Notes in Computer Science*, pages 320–334. Springer, 2015. doi:10.1007/978-3-319-32859-1_27.

36 Marcin Kubica, Tomasz Kulczynski, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. A linear time algorithm for consecutive permutation pattern matching. *Inf. Process. Lett.*, 113(12):430–433, 2013. doi:10.1016/j.ipl.2013.03.015.

37 Yoshiaki Matsuoka, Takahiro Aoki, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Generalized pattern matching and periodicity under substring consistent equivalence relations. *Theor. Comput. Sci.*, 656:225–233, 2016. doi:10.1016/j.tcs.2016.02.017.

38 Arseny M. Shur and Yulia V. Gamzova. Partial words and the interaction property of periods. *Izvestiya: Mathematics*, 68:405–428, 2004. URL: http://stacks.iop.org/1064-5632/68/i=2/a=A09.

39 Arseny M. Shur and Yulia V. Konovalova. On the periods of partial words. In Jirí Sgall, Ales Pultr, and Petr Kolman, editors, *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001 Marianske Lazne, Czech Republic, August 27-31, 2001, Proceedings*, volume 2136 of *Lecture Notes in Computer Science*, pages 657–665. Springer, 2001. doi:10.1007/3-540-44683-4_57.

40 Robert Tijdeman and Luca Zamboni. Fine and Wilf words for any periods. *Indag. Math.*, 14(1):135–147, 2003. doi:10.1016/S0019-3577(03)90076-0.

Beyond JWP: A Tractable Class of Binary VCSPs via M-Convex Intersection

Hiroshi Hirai
Department of Mathematical Informatics, University of Tokyo, Japan
hirai@mist.i.u-tokyo.ac.jp

Yuni Iwamasa
Department of Mathematical Informatics, University of Tokyo, Japan
yuni_iwamasa@mist.i.u-tokyo.ac.jp

Kazuo Murota
Department of Business Administration, Tokyo Metropolitan University, Japan.
murota@tmu.ac.jp

Stanislav Živný
Department of Computer Science, University of Oxford, United Kingdom.
standa.zivny@cs.ox.ac.uk

Abstract

A binary VCSP is a general framework for the minimization problem of a function represented as the sum of unary and binary cost functions. An important line of VCSP research is to investigate what functions can be solved in polynomial time. Cooper–Živný classified the tractability of binary VCSP instances according to the concept of "triangle," and showed that the only interesting tractable case is the one induced by the joint winner property (JWP). Recently, Iwamasa–Murota–Živný made a link between VCSP and discrete convex analysis, showing that a function satisfying the JWP can be transformed into a function represented as the sum of two M-convex functions, which can be minimized in polynomial time via an M-convex intersection algorithm if the value oracle of each M-convex function is given.

In this paper, we give an algorithmic answer to a natural question: What binary finite-valued CSP instances can be solved in polynomial time via an M-convex intersection algorithm? We solve this problem by devising a polynomial-time algorithm for obtaining a concrete form of the representation in the representable case. Our result presents a larger tractable class of binary finite-valued CSPs, which properly contains the JWP class.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics, Theory of computation → Problems, reductions and completeness, Theory of computation → Discrete optimization

Keywords and phrases Valued Constraint Satisfaction Problems, Discrete Convex Analysis, M-convexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.39

Funding The first author's research was partially supported by JSPS KAKENHI Grant Numbers 25280004, 26330023, 17K00029. The second author's research was supported by JSPS Research Fellowship for Young Scientists. The third author's research was supported by The Mitsubishi Foundation, CREST, JST, Grant Number JPMJCR14D2, Japan, and JSPS KAKENHI Grant Number 26280004. The last author's research was supported by a Royal Society University Research Fellowship. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 714532). The paper reflects only the authors' views and not the views of the ERC

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 39; pp. 39:1–39:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

or the European Commission. The European Union is not liable for any use that may be made of the information contained therein.

1 Introduction

The *valued constraint satisfaction problem (VCSP)* provides a general framework for discrete optimization (see [24] for details). Informally, the VCSP framework deals with the minimization problem of a function represented as the sum of "small" arity functions, which are called *cost functions*. It is known that various kinds of combinatorial optimization problems can be formulated in the VCSP framework. In general, the VCSP is NP-hard. An important line of research is to investigate what restrictions on classes of VCSP instances ensure polynomial time solvability. Two main types of VCSPs with restrictions are *structure-based VCSPs* and *language-based VCSPs* (see e.g., [4, 24]). Structure-based VCSPs deal with restrictions on the hypergraph structure representing the appearance of variables in a given instance. For example, Gottlob–Greco–Scarcello [7] showed that, if the hypergraph corresponding to a VCSP instance has a bounded hypertree-width, then the instance can be solved in polynomial time. Language-based VCSPs deal with restrictions on cost functions that appear in a VCSP instance. Kolmogorov–Thapper–Živný [13] gave a precise characterization of tractable valued constraint languages via the basic LP relaxation. Kolmogorov–Krokhin–Rolínek [12] gave a dichotomy for all language-based VCSPs (see also [1, 25] for a dichotomy for all language-based CSPs).

Hybrid VCSPs, which deal with a combination of structure-based and language-based restrictions, have emerged recently [4]. Among many kinds of hybrid restrictions, a *binary VCSP*, VCSP with only unary and binary cost functions, is a representative hybrid restriction that includes numerous fundamental optimization problems. Cooper–Živný [2] showed that if a given binary VCSP instance satisfies the *joint winner property (JWP)*, then it can be minimized in polynomial time. The same authors classified in [3] the tractability of binary VCSP instances according to the concept of "triangle," and showed that the only interesting tractable case is the one induced by the JWP (see also [4]). Furthermore, they introduced *cross-free convexity* as a generalization of JWP, and devised a polynomial-time minimization algorithm for cross-free convex instances F, provided a "cross-free representation" of F is given.

In this paper, we introduce a novel tractability principle going beyond triangle and cross-free representation for binary finite-valued CSPs, from now on denoted by VCSPs. A binary VCSP is formulated as follows, where $D_1, D_2, \ldots, D_r$ $(r \geq 2)$ are finite sets.

Given: Unary cost functions $F_p : D_p \to \mathbf{R}$ for $p \in \{1, 2, \ldots, r\}$ and binary cost functions $F_{pq} : D_p \times D_q \to \mathbf{R}$ for $1 \leq p < q \leq r$.

Problem: Find a minimizer of $F : D_1 \times D_2 \times \cdots \times D_r \to \mathbf{R}$ defined by

$$F(X_1, X_2, \ldots, X_r) := \sum_{1 \leq p \leq r} F_p(X_p) + \sum_{1 \leq p < q \leq r} F_{pq}(X_p, X_q). \tag{1}$$

Our tractability principle is built on *discrete convex analysis (DCA)* [18, 20], which is a theory of convex functions on discrete structures. In DCA, *L-convexity* and *M-convexity* play primary roles; the former is a generalization of submodularity, and the latter is a generalization of matroids. A variety of polynomially solvable problems in discrete optimization can be understood within the framework of L-convexity/M-convexity (see e.g., [20, 21, 22]). Recently, it has also turned out that discrete convexity is deeply linked to tractable classes of VCSPs. L-convexity is closely related to the tractability of language-based VCSPs. Various kinds of

submodularity induce tractable classes of language-based VCSP instances [13], and a larger class of such submodularity can be understood as L-convexity on certain graph structures [9]. On the other hand, Iwamasa–Murota–Živný [11] have pointed out that M-convexity plays a role in hybrid VCSPs. They revealed the reason for the tractability of a VCSP instance satisfying the JWP from a view point of M-convexity. We here continue this line of research, and explore further applications of M-convexity in hybrid VCSPs.

A function $f : \{0,1\}^n \to \mathbf{R} \cup \{+\infty\}$ is called *M-convex* [15, 20] if it satisfies the following generalization of the matroid exchange axiom: for $x = (x_1, x_2, \ldots, x_n), y = (y_1, y_2, \ldots, y_n) \in \mathrm{dom}\, f$ and $i \in \{1, 2, \ldots, n\}$ with $x_i > y_i$, there exists $j \in \{1, 2, \ldots, n\}$ with $y_j > x_j$ such that $f(x) + f(y) \geq f(x - \chi_i + \chi_j) + f(y + \chi_i - \chi_j)$, where, for a function $f : \mathcal{D} \to \mathbf{R} \cup \{+\infty\}$, the effective domain is denoted as $\mathrm{dom}\, f := \{x \in \mathcal{D} \mid f(x) < +\infty\}$, and χ_i is the ith unit vector.[4] An M-convex function can be minimized in a greedy fashion similarly to the greedy algorithm for matroids. Furthermore, a function $f : \{0,1\}^n \to \mathbf{R} \cup \{+\infty\}$ that is representable as the sum of two M-convex functions is called *M_2-convex*. As a generalization of matroid intersection, the problem of minimizing an M_2-convex function, called the *M-convex intersection problem*, can also be solved in polynomial time if the value oracle of each constituent M-convex function is given [16, 17]; see also [19, Section 5.2]. Our proposed tractable class of VCSPs is based on this result.

Let us return to binary VCSPs. The starting observation for relating VCSP to DCA is that the objective function F on $D_1 \times D_2 \times \cdots \times D_r$ can be regarded as a function f on $\{0,1\}^n$ by the following correspondence between the domains:

$$D_p := \{1, 2, \ldots, n_p\} \ni i \longleftrightarrow \underbrace{(0, \ldots, 0, \overset{i}{\check{1}}, 0, \ldots, 0)}_{n_p} \qquad (p \in \{1, 2, \ldots, r\}). \tag{2}$$

With this correspondence, the minimization of F can be transformed to that of f. A binary VCSP instance F is said to be *M_2-representable* if the function f obtained from F via the correspondence (2) is M_2-convex.

It is shown in [11] that a binary VCSP instance satisfying the JWP can be transformed to an M_2-representable instance,[5] and two M-convex summands can be obtained in polynomial time. Here the following natural question arises: *What binary VCSP instances are M_2-representable?* In this paper, we give an algorithmic answer to this question by considering the following problem:

Testing M_2-Representability

Given: A binary VCSP instance F.

Problem: Determine whether F is M_2-representable or not. If F is M_2-representable, obtain a decomposition $f = f_1 + f_2$ of the function f into two M-convex functions f_1 and f_2, where f is the function transformed from F via (2).

Our main result is the following:

▶ **Theorem 1.1.** Testing M_2-Representability *can be solved in $O(n^5)$ time.*

[4] Although M-convex functions are defined on $\mathbf{Z}^n$ in general, we only need functions on $\{0,1\}^n$ here. M-convex functions on $\{0,1\}^n$ are equivalent to the negative of *valuated matroids* introduced by Dress–Wenzel [5, 6].

[5] In [11], a binary VCSP instance satisfying the JWP was transformed into the sum of two $M^\natural$-convex functions. It can be easily seen that this function can also be transformed into the sum of two M-convex functions.

An M_2-convex function f can be minimized in polynomial time if such a decomposition can be obtained in polynomial time. Thus we obtain the following corollary of Theorem 1.1.

▶ **Corollary 1.2.** *An M_2-representable binary VCSP instance can be minimized in polynomial time.*

Our result provides us with cross-free representations, and presents a new tractable class of binary VCSPs that goes beyond JWP. A nice feature of our contribution is that the tractability based on M_2-representability is independent of a particular representation (1) of a given instance, while the tractability based on JWP or cross-free convexity depends on a representation; see the full version [10] of this paper.

Our approach to a polynomial-time algorithm for Testing M_2-Representability is outlined as follows:

- We establish a unique representation theorem of M_2-convex functions arising from binary VCSP instances (Theorem 2.2).
- With this result, our problem can be separated into two subproblems named Decomposition and Laminarization. The former is the problem of obtaining the unique representation of a given M_2-convex function, and the latter is the problem of making a laminar family from a given family of subsets by means of certain transformations.
- We devise a polynomial-time algorithm for each problem, Decomposition and Laminarization (Theorems 4.2 and 5.8).

The proofs are omitted due to space limitation. The full version [10] of this paper will give the proofs as well as more general results and application to pseudo-Boolean function optimization.

Organization. In Section 2, we introduce the representation theorem (Theorem 2.2) of quadratic M_2-convex functions arising from VCSP instances as well as the subproblems, Decomposition and Laminarization. In Sections 3 and 5, we present polynomial-time algorithms for Decomposition and Laminarization, respectively.

Notation. Let $\mathbf{Z}$, $\mathbf{R}$, $\mathbf{R}_+$, and $\mathbf{R}_{++}$ denote the sets of integers, reals, nonnegative reals, and positive reals, respectively. In this paper, functions can take the infinite value $+\infty$, where $a < +\infty$, $a + \infty = +\infty$ for $a \in \mathbf{R}$, and $0 \cdot (+\infty) = 0$. Let $\overline{\mathbf{R}} := \mathbf{R} \cup \{+\infty\}$. For a positive integer k, we define $[k] := \{1, 2, \ldots, k\}$.

2 Towards testing M_2-representability

2.1 Representation theorem

We introduce a class of quadratic functions on $\{0,1\}^n$ that has a bijective correspondence to binary VCSP instances. Let $\mathcal{A} := \{A_1, A_2, \ldots, A_r\}$ be a partition of $[n]$ with $|A_p| \geq 2$ for $p \in [r]$. We say that $f : \{0,1\}^n \to \overline{\mathbf{R}}$ is a *VCSP-quadratic function of type* $\mathcal{A}$ if f is represented as

$$f(x_1, x_2, \ldots, x_n) := \begin{cases} \sum_{i \in [n]} a_i x_i + \sum_{1 \leq i < j \leq n} a_{ij} x_i x_j & \text{if } \sum_{i \in [n]} x_i = r, \\ +\infty & \text{otherwise,} \end{cases} \tag{3}$$

where $a_i \in \mathbf{R}$ and $a_{ij} \in \overline{\mathbf{R}}$ with $a_{ij} := +\infty \Leftrightarrow i, j \in A_p$ for some $p \in [r]$. We assume $a_{ij} = a_{ji}$ for distinct $i, j \in [n]$.

Suppose that a binary VCSP instance F of the form (1) is given, where we assume $F_{pq} = F_{qp}$ for distinct $p, q \in [r]$. The transformation of F to f based on (2) in Section 1 is formalized as follows. Choose a partition $\mathcal{A} := \{A_1, A_2, \ldots, A_r\}$ of $[n]$ with $|A_p| = n_p (= |D_p|)$ and a bijective correspondence $A_p \to D_p$. Define $a_i := F_p(d)$ if $i \in A_p$ corresponds to $d \in D_p$, $a_{ij} := F_{pq}(d, e)$ if $i \in A_p$ and $j \in A_q$ correspond to $d \in D_p$ and $e \in D_q$, respectively, and $a_{ij} := +\infty$ otherwise. Then the function f in (3) is a VCSP-quadratic function of type $\mathcal{A}$.

The class of M_2-convex VCSP-quadratic functions admits a decomposition into simpler functions ℓ_X. For $X \subseteq [n]$, let $\ell_X : \{0,1\}^n \to \mathbf{R}$ be defined by

$$\ell_X(x) := \sum_{k_-(X) < k < k_+(X)} \left| k - \sum_{i \in X} x_i \right|,$$

where $k_-(X)$ is the number of indices $p \in [r]$ with $X \supseteq A_p$, and $k_+(X)$ is the number of indices $p \in [r]$ with $X \cap A_p \neq \emptyset$. That is, $\ell_X(x)$ is the sum of the distances from $x \in \{0,1\}^n$ to hyperplanes $\{x \in \mathbf{R}^n \mid \sum_{i \in X} x_i = k\}$ for $k_-(X) < k < k_+(X)$. In the following, we consider subsets X with $k_-(X) + 2 \leq k_+(X)$, and denote the family of such subsets X by

$$\Pi = \Pi_{\mathcal{A}} := \{X \subseteq [n] \mid k_-(X) + 2 \leq k_+(X)\}.$$

In other words, $X \in \Pi$ if and only if $\emptyset \neq X \cap A_p \neq A_p$ for more than one $p \in [r]$.

A family $\mathcal{F} \subseteq \Pi$ is said to be *laminar* if $X \subseteq Y$, $X \supseteq Y$, or $X \cap Y = \emptyset$ holds for all $X, Y \in \mathcal{F}$. Define $\delta_{\mathcal{A}} : \{0,1\}^n \to \overline{\mathbf{R}}$ by $\delta_{\mathcal{A}}(x) := 0$ if $\sum_{i \in A_p} x_i = 1$ for each $A_p \in \mathcal{A}$, and $\delta_{\mathcal{A}}(x) := +\infty$ otherwise. Then the following holds.

▶ **Lemma 2.1.** *For any laminar family $\mathcal{L} \subseteq \Pi$ and any positive weight $c : \mathcal{L} \to \mathbf{R}_{++}$, the function $\sum_{X \in \mathcal{L}} c(X)\ell_X$ on $\{x \in \{0,1\}^n \mid \sum_{i \in [n]} x_i = r\}$ is M-convex.*

Our representation theorem (Theorem 2.2) says that an M_2-convex VCSP-quadratic function is always represented as the sum of $\sum_{X \in \mathcal{L}} c(X)\ell_X$ on $\{x \in \{0,1\}^n \mid \sum_{i \in [n]} x_i = r\}$ and a linear function on $\mathrm{dom}\, \delta_{\mathcal{A}}$. To state it precisely, there are substantial complications to be resolved. In our setting, we are given a VCSP-quadratic function f of type $\mathcal{A}$, which is defined only on $\mathrm{dom}\, f = \mathrm{dom}\, \delta_{\mathcal{A}}$. It can happen that functions ℓ_X and ℓ_Y are identical on $\mathrm{dom}\, \delta_{\mathcal{A}}$ (i.e., $\ell_X + \delta_{\mathcal{A}} = \ell_Y + \delta_{\mathcal{A}}$) even when $X \neq Y$. Thus we have to make a judicious choice between them to demonstrate M_2-representability of f.

To cope with such complications, we define an equivalence relation $\sim$ by: $X \sim Y \Leftrightarrow \ell_X + \delta_{\mathcal{A}} = \ell_Y + \delta_{\mathcal{A}}$. For $\mathcal{F} \subseteq \Pi$, let $\mathcal{F}/\!\sim$ be the set of representatives (in $\Pi/\!\sim$) of all elements in $\mathcal{F}$. The equivalence relation is extended to subsets $\mathcal{F}, \mathcal{G}$ of Π by: $\mathcal{F} \sim \mathcal{G} \Leftrightarrow \mathcal{F}/\!\sim = \mathcal{G}/\!\sim$. A subset $\mathcal{P}$ of $\Pi/\!\sim$ is said to be *laminar* if there is a laminar family $\mathcal{L} \subseteq \Pi$ with $\mathcal{P} = \mathcal{L}/\!\sim$. A family $\mathcal{F} \subseteq \Pi$ is said to be *laminarizable* if $\mathcal{F}/\!\sim$ is laminar. For simplicity, the equivalence class of $X \in \Pi$ is also denoted by X, and a member of $\Pi/\!\sim$ is also denoted by its representative X.

Our first result is a representation theorem of M_2-convex functions.

▶ **Theorem 2.2.** *Let f be a VCSP-quadratic function of type $\mathcal{A} = \{A_1, A_2, \ldots, A_r\}$. Then f is M_2-convex if and only if there exist a laminar family $\mathcal{P}_f \subseteq \Pi/\!\sim$ and a positive weight $c_f : \mathcal{P}_f \to \mathbf{R}_{++}$ such that*

$$f = \sum_{X \in \mathcal{P}_f} c_f(X)\ell_X + \delta_{\mathcal{A}} + (\text{linear function}), \tag{4}$$

where "(linear function)" means a function $x \mapsto \sum_i p_i x_i + \alpha$ for some $(p_1, p_2, \ldots, p_n) \in \mathbf{R}^n$ and $\alpha \in \mathbf{R}$. In addition, $\mathcal{P}_f$ and c_f in (4) are uniquely determined.

By Theorem 2.2, an M_2-convex function f has the summand $f_1 := \sum_{X \in \mathcal{L}} c_f(X)\ell_X$ on $\{x \in \{0,1\}^n \mid \sum_{i \in [n]} x_i = r\}$, where $\mathcal{L}$ is a laminar family with $\mathcal{L}/\!\sim = \mathcal{P}_f$.

2.2 Decomposition and Laminarization

To test for M_2-representability by Theorem 2.2, we first solve the following problem Decomposition, which detects non-M_2-convexity of f or obtains decomposition (4).

Decomposition

Given: A VCSP-quadratic function f of type $\mathcal{A}$

Problem: Either detect the non-M_2-convexity of f, or obtain some $\mathcal{P} \subseteq \Pi/\sim$ and $c : \mathcal{P} \to \mathbf{R}_{++}$ satisfying

$$f = \sum_{X \in \mathcal{P}} c(X)\ell_X + \delta_{\mathcal{A}} + (\text{linear function}), \tag{5}$$

where $\mathcal{P}$ is not required to be laminar in general, but in case of M_2-convex f, $\mathcal{P}$ and c should coincide, respectively, with $\mathcal{P}_f$ and c_f in (4).

We emphasize that Decomposition may possibly output the decomposition (5) even when the input f is not M_2-convex, but if Decomposition detects the non-M_2-convexity then indeed the input f is not M_2-convex.

Suppose that decomposition (5) is obtained after solving Decomposition. In this case we have $\mathcal{P}$ at hand. Then we have to check for the laminarizability of an arbitrarily chosen family $\mathcal{F} \subseteq \Pi$ with $\mathcal{F}/\sim = \mathcal{P}$. This motivates us to consider the following problem.

Laminarization

Given: $\mathcal{F} \subseteq \Pi$

Problem: Determine whether there exists a laminar family $\mathcal{L}$ with $\mathcal{F} \sim \mathcal{L}$. If it exists, obtain a laminar family $\mathcal{L}$ with $\mathcal{F} \sim \mathcal{L}$.

Laminarization is a purely combinatorial problem on a set system. Indeed, the equivalence relation $\sim$ can be rephrased in a combinatorial way as follows. For $X \in \Pi$, define $\langle X \rangle := \bigcup\{A_p \in \mathcal{A} \mid \emptyset \neq X \cap A_p \neq A_p\}$, which is the union of A_p contributing to $\ell_X + \delta_{\mathcal{A}}$ nonlinearly. One can see the following.

▶ **Lemma 2.3.** *For $X, Y \in \Pi$, $X \sim Y$ if and only if $\{\langle X \rangle \cap X, \langle X \rangle \setminus X\} = \{\langle Y \rangle \cap Y, \langle Y \rangle \setminus Y\}$.*

Laminarization can be regarded as the problem of transforming a given family $\mathcal{F}$ to a laminar family by repeating the following operation: replace $X \in \mathcal{F}$ with $[n] \setminus X$, $X \cup A_p$, or $X \setminus A_p$ with some A_p satisfying $\langle X \rangle \cap A_p = \emptyset$.

A decomposition $f = f_1 + f_2$ into two M-convex functions f_1 and f_2 can be constructed from c_f and $\mathcal{L}$ found by Decomposition and Laminarization as $f_1 := \sum_{X \in \mathcal{L}} c_f(X)\ell_X$ on $\{x \in \{0,1\}^n \mid \sum_{i \in [n]} x_i = r\}$ and $f_2 := f - f_1$. By Lemma 2.1, f_1 is an M-convex function, and f_2 is a linear function on dom $\delta_{\mathcal{A}}$.

We devise an $O(n^5)$-time algorithm for Decomposition in Section 3 and an $O(n^4)$-time algorithm for Laminarization in Section 5. Thus we obtain Theorem 1.1.

▶ Remark. Our representation theorem (Theorem 2.2) and decomposition algorithm (in Section 3) are inspired by the *polyhedral split decomposition* due to Hirai [8]. This general decomposition principle decomposes, by means of polyhedral geometry, a function on a finite set $\mathcal{D}$ of points of $\mathbf{R}^n$ into a sum of simpler functions, called *split functions*, and a residue term. Actually, (5) can be viewed as a specialization of the polyhedral split decomposition, where $\mathcal{D} =$ dom $\delta_{\mathcal{A}}$, and $\ell_X + \delta_{\mathcal{A}}$ is a sum of split functions. We refer the reader to [8] for details.

3 Algorithm for Decomposition

3.1 Outline

To describe our algorithm, we need the concept of *restriction* of a VCSP-quadratic function. Let f be a VCSP-quadratic function of type $\mathcal{A} = \{A_1, A_2, \ldots, A_r\}$. For $Q \subseteq [r]$, let $\mathcal{A}_Q := \{A_p\}_{p\in Q}$ and $A_Q := \bigcup_{q\in Q} A_q$. For $Q \subseteq [r]$, the *restriction* $f_Q : \{0,1\}^{A_Q} \to \overline{\mathbf{R}}$ *of* f *to* Q is a VCSP-quadratic function of type $\mathcal{A}_Q$ defined by

$$f_Q(x) := \begin{cases} \sum_{i\in A_Q} a_i x_i + \sum_{i,j\in A_Q\ (i<j)} a_{ij} x_i x_j & \text{if } \sum_{i\in A_Q} x_i = |Q|, \\ +\infty & \text{otherwise.} \end{cases}$$

▶ **Lemma 3.1.** *If f is M_2-convex, so is the restriction f_Q for each $Q \subseteq [r]$.*

We abbreviate $\Pi_{\mathcal{A}}$ and $\Pi_{\mathcal{A}_Q}$ to Π and Π_Q, respectively. If f is M_2-convex, then f_Q can also be represented in a form similar to (4), i.e.,

$$f_Q = \sum_{X\in\mathcal{P}_{f_Q}} c_{f_Q}(X)\ell_X + \delta_{\mathcal{A}_Q} + (\text{linear function}),$$

where ℓ_X and $\delta_{\mathcal{A}_Q}$ are defined on $\{0,1\}^{A_Q}$.

Our algorithm to obtain decomposition (5) is outlined as follows, where we abbreviate $\{p,q\}$ and $\{p\}$ to pq and p, respectively, and also $\mathcal{P}_{f_{pq}}$ and $c_{f_{pq}}$ to $\mathcal{P}_{pq}$ and c_{pq}, respectively:

- We obtain a decomposition of the restriction f_Q for $Q = \{1,2\}, \{1,2,3\}, \ldots, \{1,2,3,\ldots,r\}$ in turn:
$$f_Q = \sum_{X\in\mathcal{P}_Q} c_Q(X)\ell_X + \delta_{\mathcal{A}_Q} + (\text{linear function}). \tag{6}$$
- In the initial case for $Q = \{1,2\}$, we can obtain the decomposition (6) by Algorithm 1 (Section 3.2).
- To construct the decomposition (6) for $Q = [r']$ from that for $Q = [r'-1]$, we first compute $(\mathcal{P}_{pr'}, c_{pr'})$ for all $p \in [r'-1]$ by Algorithm 1 and then, with this information, extend $(\mathcal{P}_{[r'-1]}, c_{[r'-1]})$ to $(\mathcal{P}_{[r']}, c_{[r']})$ by Algorithm 2 (Section 4).
- We perform the above extension step for $r' = 3$ to $r' = r$, to arrive at the decomposition (5) of f. This is described in Algorithm 3.

3.2 Initial case ($r = 2$)

To compute $\mathcal{P}_{pq}$ and c_{pq} for all distinct $p, q \in [r]$, we consider Decomposition algorithm for the case of $r = 2$. Namely $\mathcal{A} = \{A_1, A_2\}$. Note that $\Pi = \Pi_{\{A_1,A_2\}} = \{X \subseteq [n] \mid \emptyset \neq X \cap A_p \neq A_p \text{ for } p = 1,2\}$. A connected component with at least one edge is said to be *non-isolated*.

Note that, for distinct $L, L' \in \Pi$, we have $L \sim L' \Leftrightarrow L = [n] \setminus L'$.

▶ **Proposition 3.2.** *Algorithm 1 solves* Decomposition *in $O(n^2)$ time.*

4 General case ($r \geq 3$)

To obtain the decomposition (6) of the restriction f_Q for $Q = \{1,2\}, \{1,2,3\}, \ldots, \{1,2,3,\ldots,r\}$ in turn, we need to extend $(\mathcal{P}_{[r'-1]}, c_{[r'-1]})$ to $(\mathcal{P}_{[r']}, c_{[r']})$ with the use of $(\mathcal{P}_{pr'}, c_{pr'})$ $(p \in [r'-1])$ for $r' = 3, \ldots, r$. Algorithm 2 corresponds to this extension step.

The following proposition shows that Algorithm 2 works as expected.

Algorithm 1: (for DECOMPOSITION in the case of $r = 2$).

Input: A VCSP-quadratic function f of type $\{A_1, A_2\}$.

Step 0: Define $\alpha^* := \min_{i,j\in[n]} a_{ij}$ and $S := \{i \in [n] \mid \min_{j\in[n]} a_{ij} > \alpha^*\}$.

Step 1: For $i \in [n]$ with $b_i := \min_{j\in[n]} a_{ij} - \alpha^* > 0$, update $a_{ij} \leftarrow a_{ij} - b_i$ for $j \in [n]\setminus\{i\}$ in turn.

Step 2: Let the distinct finite values of a_{ij} $(i \in A_1,\ j \in A_2)$ be given by $\alpha_1 > \alpha_2 > \cdots > \alpha_m = \alpha^*$. For $\alpha \in \mathbf{R}$, define a graph $G_\alpha := ([n], E_\alpha)$ by $E_\alpha := \{\{i,j\} \mid i \in A_1,\ j \in A_2,\ \alpha \le a_{ij}\}$. If, for some $\alpha \in \{\alpha_1, \alpha_2, \ldots, \alpha_{m-1}\}$, a (non-isolated) connected component of G_α is not a complete bipartite graph, then output "f is not M_2-convex" and stop.

Step 3: For $s \in [m-1]$, denote by $\mathcal{L}^s$ the set of non-isolated connected components L of G_{α_s}. For $L \in \mathcal{L}^s \setminus \mathcal{L}^{s-1}$ with $s \in [m-1]$, let $\alpha_L := \alpha_s$, where $\mathcal{L}^0 := \emptyset$. Define a laminar family $\mathcal{L}$ by $\mathcal{L} := \bigcup_{s=1}^{m-1} \mathcal{L}^s$. For $L \in \mathcal{L}$, define $c : \mathcal{L} \to \mathbf{R}_{++}$ by $c(L) := (\alpha_L - \alpha_{L^+})/2$, where L^+ is the minimal element in $\mathcal{L}$ properly containing L if L is not maximal, and $\alpha_{L^+} := \alpha^*$ if L is maximal.

Step 4: Turn $c : \mathcal{L} \to \mathbf{R}_{++}$ to $\mathcal{L}/\!\sim \to \mathbf{R}_{++}$ by defining the value c on an equivalence class as the sum of $c(L)$ over (at most two) members L in the equivalence class. Output $\mathcal{P} := \mathcal{L}/\!\sim$ and c.

▶ **Proposition 4.1.** *If f is M_2-convex, and $\mathcal{P}' = \mathcal{P}_{f'}$ and $c' = c_{f'}$ hold, then $\mathcal{P} = \mathcal{P}_f$ and $c = c_f$ hold. Furthermore, Algorithm 2 runs in $O(n^4)$ time.*

Our proposed algorithm for DECOMPOSITION can be summarized as follows. It is noted that, if $\mathcal{P}$ is laminar, then $|\mathcal{P}|$ is at most $2n = 2|A_{[r]}|$ (see e.g., [23, Theorem 3.5]).

▶ **Theorem 4.2.** *Algorithm 3 solves* DECOMPOSITION *in $O(n^5)$ time.*

5 Algorithm for LAMINARIZATION

For a VCSP-quadratic function f of type $\mathcal{A}$, suppose that we have obtained $\mathcal{P} \subseteq \Pi/\!\sim$ by solving DECOMPOSITION. The next step for solving TESTING M_2-REPRESENTABILITY is to check for the laminarity of $\mathcal{P}$. Take $\mathcal{F} \subseteq \Pi$ with $\mathcal{F}/\!\sim = \mathcal{P}$; such $\mathcal{F}$ can be constructed easily from $\mathcal{P}$. The input of LAMINARIZATION is $\mathcal{F}$.

5.1 Outline

For families $\mathcal{G}, \mathcal{H} \subseteq \Pi$, we say that $\mathcal{G}$ is equivalent to $\mathcal{H}$ if $\mathcal{G} \sim \mathcal{H}$. It is easy to see that a laminar family can be constructed easily from a cross-free family $\mathcal{G}$ by switching $X \mapsto [n] \setminus X$ for appropriate $X \in \mathcal{G}$ (see e.g., [14, Section 2.2]); this can be done in $O(|\mathcal{G}|)$ time. Thus, by $X \sim [n] \setminus X$, our goal is to construct a cross-free family equivalent to the input family.

In this section, we devise a polynomial-time algorithm for constructing a desired cross-free family. Our algorithm makes use of weaker notions of cross-freeness, called 2- and 3-local cross-freeness. The existence of a cross-free family is characterized by the existence of a *2-locally cross-free family* (Section 5.2). The existence of such a 2-locally cross-free family can be checked easily by solving a 2-SAT problem. If a 2-locally cross-free family exists, a *3-locally cross-free family* also exists, and can be constructed in polynomial time (Section 5.4). From a 3-locally cross-free family, we can construct a desired cross-free family in polynomial time by the *uncrossing operations* (Section 5.3). Thus we solve LAMINARIZATION.

Algorithm 2: (for extending f' to f).

Input: A VCSP-quadratic function f of type $\mathcal{A}$ and restriction $f' := f_{[r-1]}$ given as

$$f' = \sum_{X \in \mathcal{P}'} c'(X)\ell_X + \delta_{\mathcal{A}_{[r-1]}} + (\text{linear function})$$

for a family $\mathcal{P}' \subseteq \Pi_{[r-1]}/\sim$ with $|\mathcal{P}'| \leq 2|A_{[r-1]}|$ and a positive weight c' on $\mathcal{P}'$.

Output: Either detect the non-M_2-convexity of f, or obtain expression

$$f = \sum_{X \in \mathcal{P}} c(X)\ell_X + \delta_{\mathcal{A}} + (\text{linear function}),$$

with $\mathcal{P} \subseteq \Pi/\sim$ satisfying $|\mathcal{P}| \leq 2n = 2|A_{[r]}|$ and a positive weight c on $\mathcal{P}$.

Step 1: For each $p \in [r-1]$, execute Algorithm 1 for f_{pr}. If Algorithm 1 returns "f_{pr} is not M_2-convex" for some $p \in [r-1]$, then output "f is not M_2-convex" and stop. Otherwise, obtain $\mathcal{P}_{pr}$ and c_{pr} for all $p \in [r-1]$. Let $\mathcal{P} := \emptyset$.

Step 2: If $\mathcal{P}' = \emptyset$, go to Step 3. Otherwise, do the following: Let X_0 be an element of $\mathcal{P}'$ such that $\langle X_0 \rangle$ is maximal. Let $\{p_1, p_2, \ldots, p_k\}$ be the set of indices $p \in [r-1]$ with $\langle X_0 \rangle = A_{\{p_1,p_2,\ldots,p_k\}}$. If there exist $X \in \Pi/\sim$ and $X_i \in \mathcal{P}_{p_i r}$ $(i = 1, 2, \ldots, k)$ such that $X \sim_{[r-1]} X_0$ and $X \sim_{p_i r} X_i$ for each $i \in [k]$, then go to Step 2-1. Otherwise, go to Step 2-2.

2-1: Update as

$$\begin{aligned}
&\mathcal{P} \leftarrow \mathcal{P} \cup \{X\}, \qquad c(X) \leftarrow \min\{c'(X_0), c_{p_1 r}(X_1), c_{p_2 r}(X_2), \ldots, c_{p_k r}(X_k)\},\\
&c'(X_0) \leftarrow c'(X_0) - c(X), \qquad c_{p_i r}(X_i) \leftarrow c_{p_i r}(X_i) - c(X) \quad (i \in [k]),\\
&\mathcal{P}' \leftarrow \mathcal{P}' \setminus \{X_0\} \text{ if } c'(X_0) = 0, \quad \mathcal{P}_{p_i r} \leftarrow \mathcal{P}_{p_i r} \setminus \{X_i\} \text{ if } c_{p_i r}(X_i) = 0 \quad (i \in [k]),
\end{aligned}$$

and go to Step 2.

2-2: Update as $\mathcal{P} \leftarrow \mathcal{P} \cup \{X_0\}$, $\mathcal{P}' \leftarrow \mathcal{P}' \setminus \{X_0\}$, and $c(X_0) \leftarrow c'(X_0)$. Go to Step 2.

Step 3: Update as $\mathcal{P} \leftarrow \mathcal{P} \cup \bigcup_{i \in [k]} \mathcal{P}_{p_i r}$, and $c(X) \leftarrow c_{p_i r}(X)$ for $i \in [k]$ and $X \in \mathcal{P}_{p_i r}$. Then output $\mathcal{P}$ and c.

Without loss of generality, we assume that $X \subseteq \langle X \rangle$ for every X in the input $\mathcal{F}$ and no distinct X, Y with $X \sim Y$ are contained in $\mathcal{F}$, i.e., $|\mathcal{F}| = |\mathcal{F}/\sim|$. For $X \in \mathcal{F}$, let $\overline{X} := \langle X \rangle \setminus X$; note $X \sim \overline{X}$. For $X, Y, Z \in \Pi$, we define $\langle XY \rangle := \langle X \rangle \cap \langle Y \rangle$ and $\langle XYZ \rangle := \langle X \rangle \cap \langle Y \rangle \cap \langle Z \rangle$. For $X \in \mathcal{F}$ and $Q \subseteq [r]$ with $A_Q \subseteq \langle X \rangle$, the *partition line of X on A_Q* is a bipartition $\{X \cap A_Q, \overline{X} \cap A_Q\}$ of A_Q. We also assume that $|\mathcal{F}/\sim|$ is at most $2n$ and that $X \subseteq Y$, $X \subseteq \overline{Y}$, $X \supseteq Y$, or $X \supseteq \overline{Y}$ holds on $\langle XY \rangle$ for distinct $X, Y \in \mathcal{F}$ with $\langle XY \rangle \neq \emptyset$, since otherwise $\mathcal{F}$ is not laminarizable.

We can also assume throughout that both $\langle X \rangle \setminus \langle Y \rangle$ and $\langle Y \rangle \setminus \langle X \rangle$ are nonempty for all distinct $X, Y \in \mathcal{F}$. Indeed, for each $X \in \mathcal{F}$, we add a new set A_X with $|A_X| = 2$ to the ground set $[n]$ and to the partition $\mathcal{A}$ of $[n]$; the ground set will be $[n] \cup \bigcup_{X \in \mathcal{F}} A_X$ and the partition will be $\mathcal{A} \cup \{A_X \mid X \in \mathcal{F}\}$. Define $X_+ := X \cup \{x\}$, where x is one of the two elements of A_X and $\mathcal{F}_+ := \{X_+ \mid X \in \mathcal{F}\}$. Note $\langle X_+ \rangle = \langle X \rangle \cup A_X$ and $\langle X_+ \rangle \setminus \langle Y_+ \rangle \neq \emptyset$ for all $X_+, Y_+ \in \mathcal{F}_+$. Then it is easily seen that there exists a cross-free family $\mathcal{L}$ with $\mathcal{L} \sim \mathcal{F}$ if and only if there exists a cross-free family $\mathcal{L}_+$ with $\mathcal{L}_+ \sim \mathcal{F}_+$.

Algorithm 3: (for DECOMPOSITION).

Step 1: Execute Algorithm 1 for the restriction f_{12}. If Algorithm 1 returns "f_{12} is not M_2-convex," then output "f is not M_2-convex" and stop. Otherwise, obtain $\mathcal{P}_{12}$ and c_{12}.

Step 2: For $r' = 3, \dots, r$, execute Algorithm 2 for $(f_{[r']}, \mathcal{P}_{[r'-1]}, c_{[r'-1]})$. If Algorithm 2 returns "$f_{[r']}$ is not M_2-convex" or $|\mathcal{P}_{[r']}| > 2|A_{[r']}|$ holds for some r', output "f is not M_2-convex" and stop. Otherwise, obtain $c_{[r']}$ and $\mathcal{P}_{[r']}$.

Step 3: Output $\mathcal{P} := \mathcal{P}_{[r]}$ and $c := c_{[r]}$.

5.2 2-local cross-freeness

For $A \subseteq [n]$, a pair $X, Y \subseteq [n]$ is said to be *crossing on* A if $(X \cap Y) \cap A$, $A \setminus (X \cup Y)$, $(X \setminus Y) \cap A$, and $(Y \setminus X) \cap A$ are all nonempty. A family $\mathcal{G} \subseteq \Pi$ is said to be *cross-free on* A if there is no crossing pair on A in $\mathcal{G}$. A family $\mathcal{G} \subseteq \Pi$ is called *2-locally cross-free* if no $X, Y \in \mathcal{G}$ is crossing on $\langle X \rangle \cup \langle Y \rangle$. A cross-free family is 2-locally cross-free.

The *LC-graph* $G(\mathcal{F}) = (V(\mathcal{F}), E_{\mathrm{f}} \cup E_{\mathrm{b}})$ of the input $\mathcal{F}$ is defined by

$$
\begin{aligned}
V(\mathcal{F}) &:= \{XY \mid X, Y \in \mathcal{F}, X \neq Y\},\\
E_{\mathrm{f}} &:= \{\{XY, XZ\} \mid Y \neq Z,\ (\langle Y \rangle \setminus \langle X \rangle) \cap \langle Z \rangle \neq \emptyset\},\\
E_{\mathrm{b}} &:= \{\{XY, YX\} \mid \langle XY \rangle \neq \emptyset\},
\end{aligned}
$$

where XY is an abbreviation of ordered pair (X, Y). LC stands for Local Cross-freeness. Note that the structure of LC-graph depends only on $\{\langle X \rangle \mid X \in \mathcal{F}\}$. We call an edge $e \in E_{\mathrm{f}}$ a *forward edge* and an edge $e \in E_{\mathrm{b}}$ a *backward edge*. A backward edge $e = \{XY, YX\}$ is said to be *flipping* (resp. *non-flipping*) if $X \subseteq Y$ or $X \supseteq Y$ (resp. $X \subseteq \overline{Y}$ or $X \supseteq \overline{Y}$) holds on $\langle XY \rangle$.

An *LC-labeling* is a function $s : V(\mathcal{F}) \to \{0, 1\}$ such that

$$
s(XY) = \begin{cases} s(XZ) & \text{if } \{XY, XZ\} \text{ is a forward edge,} \\ s(YX) & \text{if } \{XY, YX\} \text{ is a non-flipping backward edge,} \\ 1 - s(YX) & \text{if } \{XY, YX\} \text{ is a flipping backward edge, and} \end{cases} \tag{7}
$$

$$
(s(XY), s(YX)) = \begin{cases} (0,0) & \text{if } X \subsetneq \overline{Y}, \\ (0,1) & \text{if } X \subsetneq Y, \\ (1,0) & \text{if } X \supsetneq Y, \\ (1,1) & \text{if } X \supsetneq \overline{Y}, \end{cases} \quad \text{on } \langle XY \rangle \text{ for backward edge } \{XY, YX\}. \tag{8}
$$

Note that (8) imposes no condition if the partition lines of X and Y on $\langle XY \rangle$ are the same. Node $XY \in V(\mathcal{F})$ is said to be *fixed* if the value of an LC-labeling s for XY is determined as (8), that is, if $\langle XY \rangle \neq \emptyset$ and the partition lines of X and Y on $\langle XY \rangle$ are different.

An LC-labeling s transforms the family $\mathcal{F}$ to another family $\mathcal{F}^s$ equivalent to $\mathcal{F}$, which is given by $\mathcal{F}^s := \{X^s \mid X \in \mathcal{F}\}$ with $X^s := X \cup \bigcup\{\langle Y \rangle \setminus \langle X \rangle \mid Y \in \mathcal{F} \text{ with } s(XY) = 1\}$. Thanks to condition (7) on forward edges, we have $X^s \cap (\langle Y \rangle \setminus \langle X \rangle) = \emptyset$ for $Y \in \mathcal{F}$ with $s(XY) = 0$.

▶ **Proposition 5.1.** *There exists a 2-locally cross-free family equivalent to $\mathcal{F}$ if and only if there exists an LC-labeling s in $G(\mathcal{F})$. To be specific, $\mathcal{F}^s$ is a 2-locally cross-free family equivalent to $\mathcal{F}$.*

Algorithm 4: (for constructing a cross-free family).

Input: A 3-locally cross-free family $\mathcal{G}$.

Step 1: While there is a crossing pair X, Y in $\mathcal{G}$, apply the uncrossing operation to X, Y and modify $\mathcal{G}$ accordingly.

Step 2: Output $\mathcal{G}$.

An LC-labeling is nothing but a feasible solution for the 2-SAT problem defined by the constraints (7) and (8). Therefore we can check the existence of an LC-labeling s greedily in $O(|E_{\rm f} \cup E_{\rm b}|) = O(n^4)$ time as follows, where XY is called a *defined node* if the value of $s(XY)$ has been defined.

1. For each fixed node XY, define $s(XY)$ according to (8).
2. In each connected component of $G(\mathcal{F})$, execute a breadth-first search from a defined node XY, and define $s(ZW)$ for all reached nodes ZW according to (7). If a conflict in value assignment to $s(ZW)$ is detected during this process, output "there is no LC-labeling."
3. If there is an undefined node, choose any undefined node XY, and define $s(XY)$ as 0 or 1 arbitrarily. Then go to 2.

5.3 3-local cross-freeness

A family $\mathcal{G} \subseteq \Pi$ is called *3-locally cross-free* if $\mathcal{G}$ is 2-locally cross-free and $\{X, Y, Z\}$ is cross-free on $\langle X\rangle \cup \langle Y\rangle \cup \langle Z\rangle$ for all $X, Y, Z \in \mathcal{G}$ with $\langle XYZ\rangle \neq \emptyset$. A cross-free family is 3-locally cross-free, and a 3-locally cross-free family is 2-locally cross-free, whereas the converse is not true. We write $X \subseteq^* Y$ to mean $X \subseteq Y$ on $\langle X\rangle \cup \langle Y\rangle$.

Our objective of this subsection is to give an algorithm for constructing a desired cross-free family from a 3-locally cross-free family equivalent to the input $\mathcal{F}$. The algorithm consists of repeated applications of an elementary operation that preserves 3-local cross-freeness. The operation is defined by (9) below, and is referred to as the *uncrossing operation* to X, Y.

▶ **Proposition 5.2.** *Suppose that $\mathcal{G}$ is 3-locally cross-free. For $X, Y \in \mathcal{G}$, define*

$$\mathcal{G}' := \begin{cases} \mathcal{G} \setminus \{X, Y\} \cup \{X \cap Y, X \cup Y\} & \text{if } X \subseteq^* Y \text{ or } Y \subseteq^* X, \\ \mathcal{G} \setminus \{X, Y\} \cup \{X \setminus Y, Y \setminus X\} & \text{if } X \subseteq^* [n] \setminus Y \text{ or } [n] \setminus Y \subseteq^* X. \end{cases} \tag{9}$$

Then $\mathcal{G}'$ is a 3-locally cross-free family equivalent to $\mathcal{G}$.

Note, by the 2-local cross-freeness of $\mathcal{G}$, that all $X, Y \in \mathcal{G}$ satisfy $X \subseteq^* Y$, $Y \subseteq^* X$, $X \subseteq^* [n] \setminus Y$, or $[n] \setminus Y \subseteq^* X$. It is worth mentioning that the uncrossing operation does not preserve 2-local cross-freeness.

▶ **Proposition 5.3.** *Algorithm 4 runs in $O(n^2)$ time, and the output $\mathcal{G}$ is cross-free.*

5.4 Constructing 3-locally cross-free family

Our final goal is to show that, for the input $\mathcal{F}$ equivalent to a 2-locally cross-free family, we can always construct, in polynomial time, an LC-labeling s such that $\mathcal{F}^s$ is 3-locally cross-free. In the following, we assume the existence of an LC-labeling.

The following Lemma 5.4 indicates that, more often than not, a triple X, Y, Z in any 2-locally cross-free family is cross-free on $\langle X\rangle \cup \langle Y\rangle \cup \langle Z\rangle$.

▶ **Lemma 5.4.** *Let $\mathcal{G}$ be a 2-locally cross-free family. A triple $\{X, Y, Z\} \subseteq \mathcal{G}$ is cross-free on $\langle X\rangle \cup \langle Y\rangle \cup \langle Z\rangle$ if one of the following conditions holds:*

(1) $\langle XY\rangle \neq \emptyset$, *and* $\{X, Y\}$ *is cross-free on* $\langle X\rangle \cup \langle Y\rangle \cup \langle Z\rangle$.
(2) $\langle XY\rangle \not\subseteq \langle Z\rangle$, *and* $\langle XZ\rangle$ *or* $\langle YZ\rangle$ *is nonempty.*
(3) *The partition lines of* X, Y, Z *on* $\langle XYZ\rangle$ *are not the same.*
(4) $\langle XY\rangle = \langle ZY\rangle \neq \emptyset$, *and there is a path* $(XY, XY_1, \ldots, XY_k)$ *in* $G(\mathcal{G})$ *such that* $\{X, Y_k, Z\}$ *is cross-free on* $\langle X\rangle \cup \langle Y_k\rangle \cup \langle Z\rangle$.

To construct a 3-locally cross-free family, a particular care is needed for those triples X, Y, Z with $\langle XY\rangle = \langle YZ\rangle = \langle ZX\rangle \neq \emptyset$ for which there exists no path $(XY, XY_1, \ldots, XY_k)$ satisfying $\langle XY\rangle \neq \langle XY_k\rangle \neq \emptyset$. This motivates the notion of *special nodes* and *special connected components* in the LC-graph $G(\mathcal{F})$ defined in Section 5.2. For distinct $X, Y \in \mathcal{F}$, define

$$R(XY) := \{Z \in \mathcal{F} \mid \text{There is a path } (XY, XY_1, \ldots, XZ)\},$$
$$R^*(XY) := \{Z \in R(XY) \mid \langle XZ\rangle \neq \emptyset\}.$$

We say that XY with $\langle XY\rangle \neq \emptyset$ is *special* if $\langle XZ\rangle = \langle XY\rangle$ holds for all $Z \in R^*(XY)$. For $X, Y \in \mathcal{F}$ such that both XY and YX are special, let $v(XY)$ denote the connected component (as a set of nodes) containing XY or YX in $G(\mathcal{F})$. We call such a component *special*. Let $v^*(XY)$ denote the set of nodes ZW in $v(XY)$ with $\langle ZW\rangle \neq \emptyset$. A special component has an intriguing structure.

▶ **Proposition 5.5.** *If both* XY *and* YX *are special, then the following hold.*
(i) $v(XY) = (R^*(XY) \times R(YX)) \cup (R^*(YX) \times R(XY))$.
(ii) $v^*(XY) = (R^*(XY) \times R^*(YX)) \cup (R^*(YX) \times R^*(XY))$.
(iii) *If* $ZW \in v^*(XY)$, *then* ZW *is special and* $\langle ZW\rangle = \langle XY\rangle$.

For a special component $v = v(XY)$, we call $\langle XY\rangle$ the *center* of v; this is well-defined by (iii) of Proposition 5.5. For $Q \subseteq [r]$, the *Q-flower* is the nonempty set with size at least two of all special components having center A_Q.

▶ **Proposition 5.6.** *The Q-flower is given as* $\{v(X_iX_j) \mid 1 \leq i < j \leq p\}$ *for some* $p \geq 3$ *and distinct* $X_1, X_2, \ldots, X_p \in \mathcal{F}$ *such that* $R(X_iX_j) = R(X_{i'}X_j)$ *for all* $i, i' < j$, *and* $R(X_iX_j) \cap R(X_{i'}X_{j'}) = \emptyset$ *for all distinct* $j, j' \in [p]$, $i < j$, *and* $i' < j'$.

The above $X_1, X_2, \ldots, X_p$ are called the *representatives* of the Q-flower.

A component v is said to be *fixed* if v contains a fixed node, and said to be *free* otherwise. A special component $v(XY)$ in the Q-flower is free if and only if the partition lines of X' and Y' on A_Q are the same for all $X' \in R^*(YX)$ and $Y' \in R^*(XY)$. A *free Q-flower* is a maximal set of free components in the Q-flower such that the partition lines on A_Q is the same. Now the set of free components of the Q-flower is partitioned to free Q-flowers each of which is represented as $\{v(X_{i_s}X_{i_t}) \mid 1 \leq s < t \leq q\}$ with a subset $\{X_{i_1}X_{i_2}, \ldots, X_{i_q}\}$ of the representatives. A free Q-flower (for some $Q \subseteq [r]$) is also called a *free flower*.

We now provide a polynomial-time algorithm to construct a 3-locally cross-free family $\mathcal{F}^s$ by defining an appropriate LC-labeling s.

▶ **Proposition 5.7.** *The output* $\mathcal{F}^s$ *is 3-locally cross-free, and Algorithm 5 runs in* $O(n^4)$ *time.*

By Propositions 5.3 and 5.7, we obtain the following theorem.

▶ **Theorem 5.8.** *Algorithms 4 and 5 solve* Laminarization *in* $O(n^4)$ *time.*

Algorithm 5: (for constructing a 3-locally cross-free family).

Step 0: Determine whether there exists a 2-locally cross-free family equivalent to $\mathcal{F}$. If not, then output "$\mathcal{F}$ is not laminarizable" and stop.

Step 1: For all fixed nodes XY, define $s(XY)$ according to (8). By a breath-first search, define s on all other nodes in fixed components appropriately.

Step 2: For each component v which is free and not special, take any node XY in v. Define $s(XY)$ as 0 or 1 arbitrarily, and define $s(ZW)$ appropriately for all nodes ZW in v. Then all the remaining (undefined) components are special and free.

Step 3: For each free flower, which is assumed to be represented as $\{v(X_iX_j) \mid 1 \leq i < j \leq q\}$, do the following:

3-1: Define the value of $s(X_iX_j)$ for distinct $i, j \in [q]$ so that $\{X_1^s, X_2^s, \ldots, X_q^s\}$ is cross-free on $\bigcup_{i\in[q]}\langle X_i\rangle$.

3-2: Define $s(ZW)$ appropriately for all $ZW \in v(X_iX_j)$.

Step 4: Output $\mathcal{F}^s$.

References

1 A. A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS'17)*, pages 319–330, 2017.

2 M. C. Cooper and S. Živný. Hybrid tractability of valued constraint problems. *Artificial Intelligence*, 175:1555–1569, 2011.

3 M. C. Cooper and S. Živný. Tractable triangles and cross-free convexity in discrete optimisation. *Journal of Artificial Intelligence Research*, 44:455–490, 2012.

4 M. C. Cooper and S. Živný. *Hybrid tractable classes of constraint problems*, volume 7 of *Dagstuhl Follow-Ups Series*, chapter 4, pages 113–135. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

5 A. W. M. Dress and W. Wenzel. Valuated matroids: A new look at the greedy algorithm. *Applied Mathematics Letters*, 3(2):33–35, 1990.

6 A. W. M. Dress and W. Wenzel. Valuated matroids. *Advances in Mathematics*, 93:214–250, 1992.

7 G. Gottlob, G. Greco, and F. Scarcello. Tractable optimization problems through hypergraph-based structural restrictions. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP'09, Part II)*, pages 16–30, 2009.

8 H. Hirai. A geometric study of the split decomposition. *Discrete and Computational Geometry*, 36:331–361, 2006.

9 H. Hirai. Discrete convexity and polynomial solvability in minimum 0-extension problems. *Mathematical Programming, Series A*, 155:1–55, 2016.

10 H. Hirai, Y. Iwamasa, K. Murota, and S. Živný. A tractable class of binary VCSPs via M-convex intersection. In preparation.

11 Y. Iwamasa, K. Murota, and S. Živný. Discrete convexity in joint winner property. To appear in *Discrete Optimization*.

12 V. Kolmogorov, A. Krokhin, and M. Rolínek. The complexity of general-valued CSPs. *SIAM Journal on Computing*, 46(3):1087–1110, 2017.

13 V. Kolmogorov, J. Thapper, and S. Živný. The power of linear programming for general-valued CSPs. *SIAM Journal on Computing*, 44(1):1–36, 2015.

14 B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, Heidelberg, 5th edition, 2010.

15 K. Murota. Convexity and Steinitz's exchange property. *Advances in Mathematics*, 124:272–311, 1996.

16 K. Murota. Valuated matroid intersection, I: optimality criteria. *SIAM Journal on Discrete Mathematics*, 9:545–561, 1996.

17 K. Murota. Valuated matroid intersection, II: algorithms. *SIAM Journal on Discrete Mathematics*, 9:562–576, 1996.

18 K. Murota. Discrete convex analysis. *Mathematical Programming*, 83:313–371, 1998.

19 K. Murota. *Matrices and Matroids for Systems Analysis*. Springer, Heidelberg, 2000.

20 K. Murota. *Discrete Convex Analysis*. SIAM, Philadelphia, 2003.

21 K. Murota. Recent developments in discrete convex analysis. In W. Cook, L. Lovász, and J. Vygen, editors, *Research Trends in Combinatorial Optimization*, chapter 11, pages 219–260. Springer, Heidelberg, 2009.

22 K. Murota. Discrete convex analysis: A tool for economics and game theory. *Journal of Mechanism and Institution Design*, 1(1):151–273, 2016.

23 A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, Heidelberg, 2003.

24 S. Živný. *The Complexity of Valued Constraint Satisfaction Problems*. Springer, Heidelberg, 2012.

25 D. Zhuk. A proof of CSP dichotomy conjecture. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS'17)*, pages 331–342, 2017.

Nonuniform Reductions and NP-Completeness

John M. Hitchcock
Department of Computer Science, University of Wyoming, Laramie, USA

Hadi Shafei
Department of Mathematics and Computer Science, Northern Michigan University, Marquette, USA

Abstract

Nonuniformity is a central concept in computational complexity with powerful connections to circuit complexity and randomness. Nonuniform reductions have been used to study the isomorphism conjecture for NP and completeness for larger complexity classes. We study the power of nonuniform reductions for NP-completeness, obtaining both separations and upper bounds for nonuniform completeness vs uniform complessness in NP.

Under various hypotheses, we obtain the following separations:

1. There is a set complete for NP under nonuniform many-one reductions, but not under uniform many-one reductions. This is true even with a single bit of nonuniform advice.
2. There is a set complete for NP under nonuniform many-one reductions with polynomial-size advice, but not under uniform Turing reductions. That is, polynomial nonuniformity is stronger than a polynomial number of queries.
3. For any fixed polynomial $p(n)$, there is a set complete for NP under uniform 2-truth-table reductions, but not under nonuniform many-one reductions that use $p(n)$ advice. That is, giving a uniform reduction a second query makes it more powerful than a nonuniform reduction with fixed polynomial advice.
4. There is a set complete for NP under nonuniform many-one reductions with polynomial advice, but not under nonuniform many-one reductions with logarithmic advice. This hierarchy theorem also holds for other reducibilities, such as truth-table and Turing.

We also consider uniform upper bounds on nonuniform completeness. Hirahara (2015) showed that unconditionally every set that is complete for NP under nonuniform truth-table reductions that use logarithmic advice is also uniformly Turing-complete. We show that under a derandomization hypothesis, the same statement for truth-table reductions and truth-table completeness also holds.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Computational Complexity, NP-completeness, Reducibility, Nonuniform Complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.40

1 Introduction

Nonuniformity is a powerful concept in computational complexity. In a nonuniform computation a different algorithm or circuit may be used for each input size [31], as opposed to a uniform computation in which a single algorithm must be used for all inputs. Alternatively, nonuniform advice may be provided to a uniform algorithm – information that may not be computable by the algorithm but is computationally useful [21]. For example, nonuniformity can be used as a substitute for randomness [1]: every randomized algorithm can be replaced by a nonuniform one (BPP $\subseteq$ P/poly). It is unknown whether the same is true for NP,

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 40; pp. 40:1–40:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


but the Karp-Lipton Theorem [21] states that if the polynomial-time hierarchy does not collapse, then NP-complete problems have superpolynomial nonuniform complexity (PH is infinite implies NP $\not\subseteq$ P/poly). Hardness versus randomness tradeoffs show that such nonuniform complexity lower bounds imply derandomization (for example, EXP $\not\subseteq$ P/poly implies BPP $\subseteq$ i.o.SUBEXP [9]).

Nonuniform computation can also be used to give reductions between decision problems, when uniform reductions are lacking. The Berman-Hartmanis Isomorphism Conjecture [11] for NP asserts that all NP-complete sets are isomorphic under polynomial-time reductions. Progress towards relaxations of the Isomorphism Conjecture with nonuniform reductions has been made [2, 3, 16] under various hypotheses.

Allender et al. [5] used nonuniform reductions to investigate the complexity of sets of Kolmogorov-random strings. They showed that the sets R_{KS} and R_{Kt} are complete for PSPACE and EXP, respectively, under P/poly-truth-table reductions. R_{Kt} is not complete under polynomial-time truth-table reductions – in fact, the full polynomial-size advice is required [30].

The Minimum Circuit Size Problem (MCSP) [20] is an intriguing NP problem. It is not known to be NP-complete. Proving it is NP-complete would imply consequences we don't yet know how to prove, yet there is really no strong evidence that it isn't NP-complete. Recently Allender [4] has asked if the Minimum Circuit Size Problem [20] is NP-complete under P/poly-Turing reductions.

Buhrman et al. [13] began a systematic study of nonuniform completeness. They proved, under a strong hypothesis on NP, that every 1-tt-complete set for NP is many-one complete with 1 bit of advice. This result has been known for larger classes like EXP and NEXP without using any advice. They also proved a separation between uniform and nonuniform reductions in EXP by showing that there exists a language that is complete in EXP under many-one reductions that use one bit of advice, but is not 2-tt-complete [13]. They also proved that a nonuniform reduction can be turned into a uniform one by increasing the number of queries.

While Buhrman et al. [13] have some results about nonuniform reductions in NP, most of their results are focused on larger complexity classes like EXP. Inspired by their results on EXP, we work toward a similarly solid understanding of NP-completeness under nonuniform reductions. We give both separation and upper bound results for a variety of nonuniform and uniform completeness notions. We consider the standard polynomial-time reducibilities including many-one ($\leq_{\mathrm{m}}^{\mathrm{P}}$), truth-table ($\leq_{\mathrm{tt}}^{\mathrm{P}}$), and Turing ($\leq_{\mathrm{T}}^{\mathrm{P}}$). We will consider nonuniform reductions such as $\leq_{\mathrm{m}}^{\mathrm{P}/h(n)}$ where the algorithm computing the reduction is allowed $h(n)$ bits of advice for inputs of size n.

Separating Nonuniform Completeness from Uniform Completeness

We show in Section 3 that nonuniform reductions can be strictly more powerful than uniform reductions for NP-completeness. This is necessarily done under a hypothesis, for if P = NP, all completeness notions for NP trivially collapse. We use the Measure Hypothesis and the NP-Machine Hypothesis – two hypotheses on NP that have been used in previous work to separate NP-completeness notions [26, 28, 17]. The Measure Hypothesis asserts that NP does not have p-measure 0 [23, 25], or equivalently, that NP contains a p-random set [8, 7]. The NP-Machine Hypothesis [17] has many equivalent formulations and implies that there is an NP search problem that requires exponential time to solve almost everywhere.

We show under the Measure Hypothesis that there is a $\leq_{\mathrm{m}}^{\mathrm{P}/1}$-complete set for NP that is not $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete. In other words, nonuniform many-one reductions are stronger than

many-one reductions for NP-completeness, and this holds with even a single nonuniform advice bit.

We also show that if the nonuniform reductions are allowed more advice, we have a separation even from Turing reductions. Under the NP-Machine Hypothesis, there is a $\leq_m^{P/poly}$-complete set that is not $\leq_T^P$-complete. That is, polynomial-size advice makes a many-one reduction stronger for NP-completeness than a reduction that makes a polynomial number of adaptive queries.

Separating Uniform Completeness from Nonuniform Completeness

Next, in Section 4, we give evidence that uniform reductions may be strictly stronger than nonuniform reductions for NP-completeness.

We show under a hypothesis on NP $\cap$ coNP that adding just one more query makes a reduction more powerful than a nonuniform one for completeness: if $\mu_p(NP \cap coNP) \neq 0$, then for any $c \geq 1$, there is a $\leq_{2-tt}^P$-complete set that is not $\leq_m^{P/n^c}$-complete. This is an interesting contrast to our separation of $\leq_m^{P/poly}$-completeness from $\leq_T^P$-completeness (which includes $\leq_{2-tt}^P$-completeness). Limiting the advice on the many-one reduction to a fixed polynomial flips the separation the other way – and in fact, only two queries are needed. The $\mu_p(NP \cap coNP) \neq 0$ hypothesis is admittedly strong. However, we note that strong hypotheses on NP $\cap$ coNP have been used in some prior investigations [29, 18, 13].

Uniform Completeness Upper Bounds for Nonuniform Completeness

Despite the above separations, it is possible to replace a limited amount of nonuniformity by a uniform reduction for NP-completeness. Up to logarithmic advice may be made uniform at the expense of a polynomial number of queries:

1. A result of Hirahara [14] implies every $\leq_T^{P/\log}$-complete set for NP is also $\leq_T^P$-complete.
2. Under a derandomization hypothesis (E has a problem with high NP-oracle circuit complexity), we show that every $\leq_{tt}^{P/\log}$-complete set for NP is also $\leq_{tt}^P$-complete. The Valiant-Vazirani lemma [32] gives a randomized algorithm to reduce the satisfiability problem to the unique satisifability problem. Being able to derandomize this algorithm [22] yields a nonadaptive reduction.

These upper bound results are presented in Section 5.

Hierarchy Theorems for Nonuniform Completeness

In Section 6, we give hierarchy theorems for nonuniform NP-completeness. We separate polynomial advice from logarithmic advice: if the NP-machine hypothesis is true, then there is a $\leq_m^{P/poly}$-complete set that is not $\leq_m^{P/\log}$-complete. This also holds for other reducibilities such as truth-table and Turing.

2 Preliminaries

All languages in this paper are subsets of $\{0,1\}^*$. We use the standard enumeration of binary strings, i.e. $s_0 = \lambda, s_1 = 0, s_2 = 1, s_3 = 00, ...$ as an order on binary strings. For any language $A \subseteq \{0,1\}^*$ the characteristic sequence of A is defined as $\chi_A = A[s_0]A[s_1]A[s_2]...$ where $A[x] = 1$ or 0 depending on whether the string x belongs to A or not respectively. We identify every language with its characteristic sequence. For any binary sequence X and any string $x \in \{0,1\}^*$, $X \upharpoonright x$ is the initial segment of X for all strings before x.

We use the standard definitions of complexity classes and well-known reductions that can be found in [10, 27]. For any two languages A and B and a function $l : \mathbb{N} \to \mathbb{N}$, we say A is *nonuniform polynomial-time reducible to B with advice $l(n)$*, and we write $A \leq_{\mathrm{m}}^{\mathrm{P}/l(n)} B$, if there exists $f \in \mathrm{PF}$ and $h : \mathbb{N} \to \{0,1\}^*$ with $|h(n)| \leq l(n)$ for all n such that $(\forall x)\ x \in A \leftrightarrow f\big(x, h(|x|)\big) \in B$. The string $h(|x|)$ is called the *advice*, and it only depends on the length of the input. For a class $\mathcal{H}$ of functions mapping $\mathbb{N} \to \{0,1\}^*$, we say $A \leq_{\mathrm{m}}^{\mathrm{P}/\mathcal{H}} B$ if $A \leq_{\mathrm{m}}^{\mathrm{P}/l} B$ for some $l \in \mathcal{H}$. The class poly denotes all advice functions with length bounded by a polynomial, and log is all advice functions with length $O(\log n)$. We also use $\leq_{\mathrm{m}}^{\mathrm{P}/1}$ for a nonuniform reduction when $|h(|x|)| = 1$. Nonuniform reductions can similarly be defined with respect to other kinds of reductions like Turing, truth-table, etc.

In most of our proofs we use resource-bounded measure [23] to state our hypotheses. In the following we provide a brief description of this concept. For more details, see [23, 25, 7]. A *martingale* is a function $d : \{0,1\}^* \to [0,\infty)$ where $d(\lambda) > 0$ and $\forall x \in \{0,1\}^*$, $2d(x) = d(x0) + d(x1)$. We say a martingale *succeeds* on a set $A \subseteq \{0,1\}^*$ if $\limsup_{n\to\infty} d(A \upharpoonright n) = \infty$, where $A \upharpoonright n$ is the length n prefix of A's characteristic sequence. One can think of the martingale d as a strategy for betting on the consecutive bits of the characteristic sequence of A. The martingale is allowed to use the first $n-1$ bits of A when betting on the n^{th} bit. Betting starts with the initial capital $d(\lambda) > 0$, and $d(A \upharpoonright n-1)$ denotes the capital after betting on the first $(n-1)$ bits. At this stage the martingale bets some amount a where $0 \leq a \leq d(A \upharpoonright n-1)$ that the next bit is 0 and the rest of the capital, i.e. $d(A \upharpoonright n-1) - a$, that the next bit is 1. If the n^{th} bit is 0, then $d(A \upharpoonright n) = 2a$. Otherwise, $d(A \upharpoonright n) = 2(d(A \upharpoonright n-1) - a)$. For any time bound $t(n)$, we say a language L is $t(n)$-random if no $O(t(n))$-computable martingale succeeds on L. A language is p-random if it is n^c-random for every c. A language is p_2-random if it is $2^{\log^c n}$-random for some c. A class of languages C has p-measure 0, written $\mu_{\mathrm{p}}(C) = 0$, if there is a c such that no language in C is n^c-random. Similarly, C has p_2-measure 0, written $\mu_{\mathrm{p}_2}(C) = 0$, if there is a c such that no language in C is $2^{\log^c n}$-random. If C is closed under $\leq_{\mathrm{m}}^{\mathrm{P}}$-reductions, then $\mu_{\mathrm{p}}(C) = 0$ if and only if $\mu_{\mathrm{p}_2}(C) = 0$ [19].

We will use the *Measure Hypothesis* that $\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0$ and the NP-*Machine Hypothesis* [17]: there is an NP machine M and an $\epsilon > 0$ such that M accepts 0^* and no 2^{n^ϵ}-time-bounded Turing machine computes infinitely many accepting computations of M. The Measure Hypothesis implies the NP-Machine Hypothesis [17].

3 Separating Nonuniform Completeness from Uniform Completeness

Our first theorem separates nonuniform many-one completeness with one bit of advice from uniform many-one completeness for NP, under the measure hypothesis. Buhrman et al. [13] proved the same result for EXP unconditionally.

▶ **Theorem 1.** *If $\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0$ then there exists a set $D \in \mathrm{NP}$ that is* NP*-complete with respect to $\leq_{\mathrm{m}}^{\mathrm{P}/1}$-reductions but is not $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete.*

Proof. Let $R \in \mathrm{NP}$ be p-random. We use R and SAT to construct the following set:

$$D = \begin{array}{l} \langle \phi, 0\rangle : \phi \in \mathrm{SAT} \ \vee\ 0^{|\phi|} \in R\rangle \\ \bigcup \langle \phi, 1\rangle : \phi \in \mathrm{SAT} \ \wedge\ 0^{|\phi|} \in R\rangle \end{array}$$

It follows from closure properties of NP that $D \in \mathrm{NP}$. It is also easy to see that $\mathrm{SAT} \leq_{\mathrm{m}}^{\mathrm{P}/1} D$ via $\phi \to \langle \phi, R[0^{|\phi|}]\rangle$. Note that $R[0^{|\phi|}]$ is one bit of advice, and it is 1 or 0 depending on whether or not $0^{|\phi|} \in R$. We will prove that D is not $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for NP. To get a

contradiction, assume that D is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete. Therefore SAT $\leq^{\mathrm{P}}_{\mathrm{m}}$ D via some polynomial-time computable function f. Then $(\forall\phi)\ \phi \in \mathrm{SAT} \leftrightarrow f(\phi) \in D$. Based on the value of SAT$[\phi]$ and the second component of $f(\phi)$ we consider four cases:

1. $\phi \in \mathrm{SAT}\ \wedge f(\phi) = \langle\psi, 0\rangle$, for some formula ψ.
2. $\phi \notin \mathrm{SAT}\ \wedge f(\phi) = \langle\psi, 0\rangle$, for some formula ψ.
3. $\phi \in \mathrm{SAT}\ \wedge f(\phi) = \langle\psi, 1\rangle$, for some formula ψ.
4. $\phi \notin \mathrm{SAT}\ \wedge f(\phi) = \langle\psi, 1\rangle$, for some formula ψ.

In the second case above we have $\mathrm{SAT}[\phi] = \mathrm{SAT}[\psi] \vee R[0^{|\psi|}]$ and $\phi \notin \mathrm{SAT}$. Therefore $\mathrm{SAT}[\psi] \vee R[0^{|\psi|}] = 0$ which implies $R[0^{|\psi|}] = 0$. Consider the situation where the second case happens and $|\psi| \geq |\phi|/2$. The following argument shows that $R[0^{|\psi|}]$ is computable in $2^{5|\psi|}$ time in this situation. We apply f to every string of length at most $2|\psi|$, looking for a formula ϕ of length at most $2|\psi|$ such that $f(\phi) = \langle\psi, 0\rangle$ and $\phi \notin \mathrm{SAT}$. We are applying f which is computable in polynomial time to at most $2^{2|\psi|+1}$ strings. This can be done in $2^{3|\psi|}$ steps. Checking if $\phi \notin \mathrm{SAT}$ can be done in at most $2^{2|\psi|}$ steps for each ϕ. Therefore the whole algorithm takes at most $2^{5|\psi|}$ steps to terminate. If this case happens for infinitely many ψ's we will have a polynomial-time martingale that succeeds on R which contradicts the p-randomness of R. As a result, there cannot be infinitely many ϕ's that $\phi \notin \mathrm{SAT}$, $f(\phi) = \langle\psi, 0\rangle$, and $|\psi| \geq |\phi|/2$. This is because if there are infinitely many such ϕ's, then there must be infinitely many n's such that for each n there exists a ϕ satisfying the above properties. Since we assumed $|\psi| \geq |\phi|/2$ it follows that there must be infinitely many such ψ's, but we proved that this cannot happen.

An analagous argument for the third case there cannot be infinitely many ϕ's that $\phi \notin \mathrm{SAT}$, $f(\phi) = \langle\psi, 0\rangle$, and $|\psi| \geq |\phi|/2$. Therefore we have:

1. For almost every ϕ, if $\phi \notin \mathrm{SAT}\ \wedge f(\phi) = \langle\psi, 0\rangle$, then $|\psi| < |\phi|/2$.
2. For almost every ϕ, if $\phi \in \mathrm{SAT}\ \wedge f(\phi) = \langle\psi, 1\rangle$, then $|\psi| < |\phi|/2$.

It follows from these two facts that for almost every ϕ, if $|\psi| \geq |\phi|/2$, then SAT$[\phi]$ can be computed in polynomial time:

1. If $f(\phi) = \langle\psi, 0\rangle$ and $|\psi| \geq |\phi|/2$, then $\phi \in \mathrm{SAT}$.
2. If $f(\phi) = \langle\psi, 1\rangle$ and $|\psi| \geq |\phi|/2$, then $\phi \notin \mathrm{SAT}$.

Note that the only computation required in the algorithm above is computing f on ϕ which can be done in polynomial time. To summarize, for every formula ϕ it is either the case that when we apply f to ϕ the new formula ψ satisfies $|\psi| < |\phi|/2$ or SAT$[\phi]$ is computable in polynomial time. In the following we use this fact and the many-one reduction from SAT to D to introduce a $(\log n)$-tt-reduction from SAT to R.

The many-one reduction from SAT to D implies that $(\forall\phi)\ \phi \in \mathrm{SAT} \leftrightarrow f(\phi) \in D$. In other words:

$$(\forall\phi)\ f(\phi) = \langle\psi_1, i\rangle \ \text{ and } \ \mathrm{SAT}[\phi] = \mathrm{SAT}[\psi_1] \diamond_1 R[0^{|\psi_1|}] \tag{1}$$

where $\diamond_1$ is $\vee$ or $\wedge$ when $i = 0$ or 1 respectively.

Fix two strings a and b such that $a \in R$ and $b \notin R$. If $|\psi_1| \geq |\phi|/2$ then SAT$[\phi]$ is computable in polynomial time, and our reduction maps ϕ to either a or b depending on SAT$[\phi]$ being 1 or 0 respectively. To put it differently, the right hand side of (1) will be substituted by $R[a]$ or $R[b]$ respectively.

On the other hand, if $|\psi_1| < |\phi|/2$ then we repeat the same process for ψ_1. We apply f to ψ_1 to get

$$\mathrm{SAT}[\psi_1] = \mathrm{SAT}[\psi_2] \diamond_2 R[0^{|\psi_2|}] \tag{2}$$

By substituting this in (1) we will have:

$$\mathrm{SAT}[\phi] = (\mathrm{SAT}[\psi_2] \diamond_2 R[0^{|\psi_2|}]) \diamond_1 R[0^{|\psi_1|}] \tag{3}$$

Again, if $|\psi_2| \geq |\psi_1|/2$ then $\mathrm{SAT}[\psi_1]$ is computable in polynomial time, and its value can be substituted in (2) to get a reduction from SAT to R. On the other hand, if $|\psi_2| < |\psi_1|/2$ then we use f again to find ψ_3 such that:

$$\mathrm{SAT}[\psi_2] = \mathrm{SAT}[\psi_3] \diamond_3 R[0^{|\psi_3|}] \tag{4}$$

By substituting this in (3) we will have:

$$\mathrm{SAT}[\phi] = \big((\mathrm{SAT}[\psi_3] \diamond_3 R[0^{|\psi_3|}]) \diamond_2 R[0^{|\psi_2|}]\big) \diamond_1 R[0^{|\psi_1|}] \tag{5}$$

We repeat this process up to $(\log n)$ times where $n = |\phi|$. If there exists some $i \leq (\log n)$ such that $|\psi_{i+1}| \geq |\psi_i|/2$, then we can compute $\mathrm{SAT}[\psi_i]$ in polynomial time and substitute its value in the following equation:

$$\mathrm{SAT}[\phi] = \big((\mathrm{SAT}[\psi_i] \diamond_k R[0^{|\psi_i|}]) \diamond_{i-1} R[0^{|\psi_{i-1}|}]\big) \ldots \diamond_1 R[0^{|\psi_1|}] \tag{6}$$

This gives us an i-tt-reduction from SAT to R for some $i < (\log n)$.

On the other hand, if $|\psi_{i+1}| < |\psi_i|/2$ for every $i \leq (\log n)$ then we will have:

$$\mathrm{SAT}[\phi] = \big((\mathrm{SAT}[\psi_{(\log n)}] \diamond_{(\log n)} R[0^{|\psi_{(\log n)}|}]) \diamond_{(\log n)-1} R[0^{|\psi_{(\log n)-1}|}]\big) \ldots \diamond_1 R[0^{|\psi_1|}] \tag{7}$$

It follows from the construction that the length of ψ_i's is halved on each step. Therefore $|\psi_{(\log n)}|$ must be constant in n. As a result $\mathrm{SAT}[\psi_{(\log n)}]$ is computable in constant time. If we compute the value of $\mathrm{SAT}[\psi_{(\log n)}]$, and substitute it in (7) we will have a $(\log n)$-tt-reduction from SAT to R. In any case, we have shown that if SAT is many-one reducible to D, we can use this reduction to define a polynomial time computable $(\log n)$-tt-reduction from SAT to R. This means that R is $(\log n)$-tt-complete for NP. Buhrman and van Melkebeek [12] showed that complete sets for NP under $\leq^{\mathrm{P}}_{n^\alpha - \mathrm{tt}}$-reductions have p_2-measure 0. Since this complete degree is closed under $\leq^{\mathrm{P}}_{\mathrm{m}}$-reductions, it also has p-measure 0 [19]. Therefore the $(\log n)$-tt-completeness of R contradicts its p-randomness, which completes the proof. ◀

This next theorem is based on a result of Hitchcock and Pavan [16] that separated strong nondeterministic completeness from Turing completeness for NP. We separate nonuniform many-one completeness with polynomial advice from Turing completeness.

▶ **Theorem 2.** *If the* NP*-machine hypothesis holds, then there exists a* $\leq^{\mathrm{P/poly}}_{\mathrm{m}}$*-complete set in* NP *that is not* $\leq^{\mathrm{P}}_{\mathrm{T}}$*-complete.*

Proof. We follow the setup in [16]. Assume the NP-machine hypothesis holds. Then it can be shown there exists an NP-machine M that accepts 0^* such that no 2^{n^3}-time bounded Turing machine can compute infinitely many of its computations. Consider the following NP set:

$$A = \{\langle \phi, a\rangle \mid \phi \in \mathrm{SAT} \text{ and } a \text{ is an accepting computation of } M(0^{|\phi|})\} \tag{8}$$

The mapping $\phi \to \langle \phi, a\rangle$ where a is the first accepting computation of $M(0^{|\phi|})$ is a $\leq^{\mathrm{P/poly}}_{\mathrm{m}}$-reduction from SAT to A. Note that a only depends on the length of ϕ and $|a|$ is polynomial in the $|\phi|$. Therefore A is $\leq^{\mathrm{P/poly}}_{\mathrm{m}}$-complete for NP. It is proved in [16] that A is not $\leq^{\mathrm{P}}_{\mathrm{T}}$-complete. ◀

Because the measure hypothesis implies the NP-machine hypothesis, we have the following corollary.

▶ **Corollary 3.** *If* $\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0$*, then there exists a* $\leq^{\mathrm{P/poly}}_{\mathrm{m}}$*-complete set in* NP *that is not* $\leq^{\mathrm{P}}_{\mathrm{T}}$*-complete.*

4 Separating Uniform Completeness from Nonuniform Completeness

Buhrman et al. [13] showed there is a $\leq^{\mathrm{P}}_{2-\mathrm{tt}}$-complete set for EXP that is not $\leq^{\mathrm{P}/1}_{\mathrm{m}}$-complete. We show the same for NP-completeness under a strong hypothesis on NP ∩ coNP; in fact, the set is not even complete with many-one reductions that use a fixed polynomial amount of advice. In the proof, we use the construction of a $\leq^{\mathrm{P}}_{2-\mathrm{tt}}$-complete set that was previously used to separate $\leq^{\mathrm{P}}_{2-\mathrm{tt}}$-completeness from $\leq^{\mathrm{P}}_{1-\mathrm{tt}}$-completeness [29] and $\leq^{\mathrm{P}}_{2-\mathrm{tt}}$-autoreducibility from $\leq^{\mathrm{P}}_{1-\mathrm{tt}}$-autoredicibility [18].

▶ **Theorem 4.** *If $\mu_{\mathrm{p}}(\mathrm{NP} \cap \mathrm{coNP}) \neq 0$ then for every $c \geq 1$, there exists a set $A \in \mathrm{NP}$ that is $\leq^{\mathrm{P}}_{2-\mathrm{tt}}$-complete but is not $\leq^{\mathrm{P}/n^c}_{\mathrm{m}}$-complete.*

Proof. We know that $\mu_{\mathrm{p}}(\mathrm{NP} \cap \mathrm{coNP}) \neq 0$ implies $\mu_{\mathrm{p}_2}(\mathrm{NP} \cap \mathrm{coNP}) \neq 0$ [19]. Therefore we can assume there exists $R \in \mathrm{NP} \cap \mathrm{coNP}$ that is p_2-random. We fix $c \geq 1$, and define $A = 0(R \cap \mathrm{SAT}) \cup 1(\bar{R} \cap \mathrm{SAT})$, where $\bar{R}$ is R's complement. It follows from closure properties of NP that $A \in \mathrm{NP}$. We can define a polynomial-time computable 2-tt-reduction from SAT to A as follows: on input x we make two queries $0x$ and $1x$ from A, and we have $x \in \mathrm{SAT} \leftrightarrow (0x \in A \vee 1x \in A)$. Therefore A is $\leq^{\mathrm{P}}_{2-\mathrm{tt}}$-complete in NP. We will show that A is not $\leq^{\mathrm{P}/n^c}_{\mathrm{m}}$-complete. To get a contradiction, assume A is $\leq^{\mathrm{P}/n^c}_{\mathrm{m}}$-complete in NP. This implies that $R \leq^{\mathrm{P}/n^c}_{\mathrm{m}} A$ via functions $f \in \mathrm{PF}$ and $h : \mathbb{N} \to \{0,1\}^*$ where $(\forall n)\ |h(n)| = n^c$. In other words:

$$(\forall x) R[x] = A[f\big(x, h(|x|)\big)] \ \text{ where } |h(n)| = n^c \tag{9}$$

For each length n the advice a_n has length n^c. As a result, there are 2^{n^c} possibilities for a_n. For each length n we define 2^{n^c} martingales such that each martingale assumes one of these possible strings is the actual advice for length n, and uses (9) to bet on R. We divide the capital into 2^{n^c} equal shares between these martingales. In the worst case, the martingales that do not use the right advice lose their share of the capital. We define these martingales such that the martingale that uses the right advice multiplies its share by 2^{n^c+1}. We will also show that this happens for infinitely many lengths n, which gives us a p_2-strategy to succeed on R. Note that based on the argument above, we can only focus on the martingale that uses the right advice for each length. To say it differently, in the rest of the proof we assume that we know the right advice for each length, but the price that we have to pay is to show that our martingale can multiply its capital by 2^{n^c+1}.

For each length n we first compute SAT[z] for every string z of length n. In particular, we are interested in the following set:

$$A_n = \{z \mid |z| = n \text{ and } z \notin \mathrm{SAT}\}$$

If $|A_n| < n^{2c}$ we do not bet on any string of length n. It follows from paddability of SAT that there must be infinitely many n's such that $|A_n| \geq n^{2c}$. Assume n is a length where $|A_n| \geq n^{2c}$, and let a_n be the right advice for length n. For any string x, let $v(0x) = v(1x) = x$. Consider the following set:

$$C_n = \{z \mid \ |z| = n,\ z \notin \mathrm{SAT},\ \text{and } v\big(f(z, a_n)\big) > z\}$$

▶ **Claim 5.** *There must be infinitely many n's where $|A_n| \geq n^{2c}$ and $|C_n| \geq n^{2c} - n^c$.*

Proof. Assume the claim does not hold. Then we have: $(\forall^\infty n)\ |A_n| \geq n^{2c} \to |C_n| < n^{2c} - n^c$. This means for almost every n if $|A_n| \geq n^{2c}$ then there are $n^c + 1$ strings of length n, $z_1, z_2, ..., z_{n^c+1}$, satisfying the following property:

$$(\forall\ 1 \leq i \leq n^c + 1)\ R[z_i] = A[f(z_i, a_n)] \ \wedge\ v(f(z_i, a_n)) \leq z_i \tag{10}$$

It follows from the definition of A that $A[y] = \tilde{R} \cap \mathrm{SAT}[v(y)]$ where $\tilde{R}$ is R or $\bar{R}$ depending on whether y starts with a 0 or 1 respectively. Therefore (10) turns into:

$$(\forall\, 1 \leq i \leq n^c + 1)\ R[z_i] = (\tilde{R} \cap \mathrm{SAT})[v(f(z_i, a_n))]\ \wedge\ v(f(z_i, a_n)) \leq z_i \tag{11}$$

We use (11) to define a martingale that predicts $R[z_i]$ for every $1 \leq i \leq n^c + 1$. Since we know $R[z_i] = \tilde{R}[v(f(z_i, a_n))] \wedge \mathrm{SAT}[v(f(z_i, a_n))]$ our martingale computes $\tilde{R}[v(f(z_i, a_n))] \wedge \mathrm{SAT}[v(f(z_i, a_n))]$ and bets on $R[z_i]$ and $\tilde{R}[v(f(z_i, a_n))] \wedge \mathrm{SAT}[v(f(z_i, a_n))]$ having the same value. Now we need to show why a polynomial time martingale has enough time to compute $\tilde{R}[v(f(z_i, a_n))] \wedge \mathrm{SAT}[v(f(z_i, a_n))]$. Note that we know $v(f(z_i, a_n)) \leq z_i$ so it is either the case that $v(f(z_i, a_n)) < z_i$ or $v(f(z_i, a_n)) = z_i$. In the first case, the martingale has access to $\tilde{R}[v(f(z_i, a_n))]$, and has enough time to compute $\mathrm{SAT}[v(f(z_i, a_n))]$. In the second case we know that $\mathrm{SAT}[v(f(z_i, a_n))] = 0$ therefore $R[z_i] = 0$. This implies that we can double the capital for each z_i. As a result, the capital can be multiplied by 2^{n^c+1}. If this happens for infinitely many n's we have a martingale that succeeds on R which is a contradiction. This completes the proof of Claim 5. ◀

The following claim states that when applying f to elements of C_n there cannot be many collisions. Define:

$$D_n = \{z \in C_n \mid (\exists\, y \in C_n)\ y < z\ \wedge\ f(y, a_n) = f(z, a_n)\}$$

▶ **Claim 6.** *There cannot be infinitely many n's such that $|D_n| \geq n^c + 1$.*

Proof. To get a contradiction, assume there are infinitely many n's such that $|D_n| \geq n^c + 1$. Let $t_1, t_2, ..., t_{n^c+1}$ be the first such strings. Then we have:

$$(\forall\, 1 \leq i \leq n^c + 1)\ (\exists\, r_i)\ r_i \in C_n\ \wedge\ r_i < t_i\ \wedge\ f(r_i, a_n) = f(t_i, a_n)$$

It follows that:

$$(\forall\, 1 \leq i \leq n^c + 1)\ (\exists\, r_i)\ r_i \in D_n\ \wedge\ r_i < t_i\ \wedge\ R[r_i] = R[t_i]$$

We can define a martingale that looks up the value of $R[r_i]$, and bets on $R[t_i]$ based on the equation above. This means that we can double the capital by betting on $R[t_i]$ for every $1 \leq i \leq n^c + 1$. As a result, the capital will be multiplied by 2^{n^c+1}. If this happens for infinitely many n's we will have a martingale that succeeds on R which is a contradiction. This completes the proof of Claim 6. ◀

Assume n is a length where $|C_n| \geq n^{2c} - n^c$. We have shown that there are infinitely many such n's. We claim that for infinitely many of these n's, since R is p_2-random, there must be at least $(n^{2c} - n^c)/4$ strings in C_n that also belong to R.

▶ **Claim 7.** $(\forall^\infty n)\ |C_n| \geq (n^{2c} - n^c)\ \rightarrow\ |C_n \cap R| \geq (n^{2c} - n^c)/4$.

Proof. Assume this claim does not hold. Then we have:

$$(\exists^\infty n)\ |C_n| \geq n^{2c} - n^c\ \wedge\ |C_n \cap R| < (n^{2c} - n^c)/4$$

We use this assumption to define a polynomial time martingale that succeeds on R. We divide the original capital such that the martingale has $1/2n^2$ of the original capital for each length. Note that finding n's where $|C_n| \geq n^{2c} - n^c$ consists of computing SAT for every string of length n, and counting the number of negative answers, which can be done in

at most 2^{3n} steps, followed by applying f to these strings and comparing $v(f(z, a_n))$ and z, which can be done in time at most 2^{2n}. This means a polynomial-time martingale has enough time to detect C_n's where $|C_n| \geq n^{2c} - n^c$. After detecting these C_n's we use a simple martingale that for every string z in C_n bets 2/3 of the capital on $R[z] = 0$ and the rest on $R[z] = 1$. It is easy to verify that in the cases where $|C_n \cap R| < (n^{2c} - n^c)/4$ we win enough so the martingale succeeds on R. This completes the proof of Claim 7. ◀

Let n be a length where $|C_n \cap R| \geq (n^{2c} - n^c)/4$, and consider the image of $C_n \cap R$ under $f(\cdot, a_n)$:

$$I_n = \{f(z, a_n) \mid z \in C_n \cap R\}$$

It follows from Claim 6 that $|I_n| \geq [(n^{2c} - n^c)/4] - n^c$. If we consider the image of I_n under $v(\cdot)$ we have:

$$V_n = \{v(f(z, a_n)) \mid z \in C_n \cap R\}$$

It is easy to see that $|V_n| \geq |I_n|/2$. Therefore for large enough n we have $|V_n| \geq n^c + 1$. Now if we use (9) we have $R[z] = \tilde{R} \cap \mathrm{SAT}[v(f(z, a_n))]$. We know that $z \in R$. This implies that $\tilde{R}[v(f(z, a_n))] = 1$. Therefore a martingale that bets on $\tilde{R}[v(f(z, a_n))] = 1$ can double the capital each time. Since $|V_n| \geq n^c + 1$ this martingale multiplies the capital by 2^{n^c+1}. As a result, we have a martingale that succeeds on R, which completes the proof. ◀

5 Uniform Upper Bounds on Nonuniform Completeness

In this section, we consider whether nonuniformity can be removed in NP-completeness, at the expense of more queries.

Buhrman et al. [13] proved that every $\leq_{\mathrm{T}}^{\mathrm{P}/\log}$-complete set for EXP is also $\leq_{\mathrm{T}}^{\mathrm{P}}$-complete using a tableaux method. Hirahara [14] proved a more general result that implies the same for NP.

▶ **Theorem 8.** (Hirahara [14]) *Every $\leq_{\mathrm{T}}^{\mathrm{P}/\log}$-complete set in* NP *is $\leq_{\mathrm{T}}^{\mathrm{P}}$-complete.*

Valiant and Vazirani [32] proved that there exists a randomized polynomial-time algorithm such that given any formula ϕ, outputs a list of formulas l such that:

1. Every assignment that satisfies a formula in l also satisfies ϕ.
2. If ϕ is satisfiable, then with high probability at least one of the formulas in l is uniquely satisfiable.

Klivans and van Melkebeek [22] showed that Valiant-Vazirani lemma can be derandomized if E^{NP} contains a problem with exponential NP-oracle circuit complexity. This yields a deterministic polynomial-time algorithm that given any ϕ, outputs a list of formulas l such that:

1. Every assignment that satisfies a formula in l also satisfies ϕ.
2. If ϕ is satisfiable, then one of the formulas in l is uniquely satisfiable.

▶ **Theorem 9.** *If E^{NP} contains a problem with* NP*-oracle circuit complexity $2^{\Omega(n)}$, then every $\leq_{\mathrm{tt}}^{\mathrm{P}/\log}$-complete set in* NP *is $\leq_{\mathrm{tt}}^{\mathrm{P}}$-complete.*

Proof. Let A be an arbitrary $\leq_{\mathrm{m}}^{\mathrm{P}/1}$-complete set in NP. This case includes most of the important details and makes describing the proof simpler. We will extend to $\leq_{\mathrm{tt}}^{\mathrm{P}/\log}$ case later. We will define a $\leq_{\mathrm{tt}}^{\mathrm{P}}$-reduction from SAT to A.

We define a padded version of SAT as follows:

$\widehat{\text{SAT}} = \{\phi 10^n \mid n \in \mathbb{N} \text{ and } \phi \in \text{SAT}\}$

Then $\widehat{\text{SAT}} \in \text{NP}$, so $\widehat{\text{SAT}} \leq_{\text{m}}^{\text{P}/1} A$ via some $f \in \text{PF}$ and some $h : \mathbb{N} \to \{0,1\}$ where $(\forall \phi)\ \widehat{\text{SAT}}[\phi] = A[f(\phi, h(|\phi|))]$.

We will use $\widehat{\text{SAT}}$ to pad formulas that have different lengths, and make them of the same length. Fix an input formula ϕ over n Boolean variables x_1,...,x_n, and let $m \in \mathbb{N}$ be large enough such that all formulas $\phi \wedge x_1$, $\phi \wedge \neg x_1$, $\phi \wedge x_1 \wedge x_2$, ..., and $\phi \wedge \neg x_1 \wedge \neg x_2 \wedge \cdots \wedge \neg x_n$ can be padded into formulas of length m. We denote the padded version of these formulas by putting a bar on them. For example, the padded version of $\phi \wedge x_1$ is denoted by $\overline{\phi \wedge x_1}$.

Before describing the rest of the algorithm, observe that the process of reducing search to decision for a Boolean formula can be done using independent queries in the case that the formula is uniquely satisfiable. This is due to the fact that if a formula $\psi(y_1, \ldots, y_m)$ is uniquely satisfiable, then for each $1 \leq j \leq m$ exactly one of the formulas $\psi \wedge x_j$ and $\psi \wedge \neg x_j$ is satisfiable. Therefore the unique satisfying assignment can be found by making m independent queries to SAT, i.e. $\psi \wedge x_1, \ldots, \psi \wedge x_m$.

Using the hypothesis to derandomize the Valiant-Vazirani algorithm [22], we have a deterministic algorithm that on input $\phi(x_1, \ldots, x_n)$ outputs a list containing polynomially many formulas ψ_1,...,ψ_m satisfying properties described above. For each formula $\psi_j(y_1^j, ..., y_{n_j}^j)$ consider $\psi_j \wedge y_k^j$'s for every $1 \leq k \leq n_j$, and use padding in $\widehat{\text{SAT}}$ to turn these formulas into formulas of the same length. We denote the padded version of $\psi_j \wedge y_k^j$ by ψ_j^k for simplicity. For each ψ_j we make n_j independent queries to A: $q_1^j = f(\psi_j^1, 0), \ldots, q_{n_j}^j = f(\psi_j^{n_j}, 0)$. For each one of these queries if the answer is positive we set the respective variable to 1 and 0 otherwise. We repeat this process using 1 as advice, and we will have $2m$ assignments. We argue that ϕ is satisfiable if and only if at least one of these assignments satisfies it. If ϕ is not satisfiable then obviously none of these assignments will satisfy it. On the other hand, if $\phi \in \text{SAT}$ then at least one of the ψ_j's must be uniquely satisfiable. In this case the process described above will find this unique satisfying assignment. Again, by the Valiant-Vazirani lemma we know that every assignment that satisfies at least one of the ψ_j's must also satisfy ϕ, which means one of the $2m$ assignments produced by the algorithm above will satisfy ϕ in the case that ϕ is satisfiable. It is evident from the algorithm that the queries are independent. It is also easy to see that the reduction runs in polynomial time in $|\phi|$ since we are applying a polynomial-time computable function f to arguments about the same length as ϕ, and we are doing this $2m$ times which is polynomial in $|\phi|$. Therefore this algorithm defines a polynomial-time truth-table reduction from SAT to A.

If the nonuniform reduction in the theorem above uses k bits of advice instead of considering two cases in the proof there are 2^k cases to be considered. If $k \in O(\log n)$ then this can be done in polynomial time. Also note that the nonuniform reduction can be a truth-table reduction instead of a many-one reduction, and the same proof still works. ◀

The measure hypothesis on NP implies that E^{NP} has high NP-oracle circuit complexity [6, 24, 15]. Therefore we have the following.

▶ **Corollary 10.** *If $\mu_{\text{p}}(\text{NP}) \neq 0$, then every $\leq_{\text{tt}}^{\text{P}/\log}$-complete set in* NP *is $\leq_{\text{tt}}^{\text{P}}$-complete.*

6 Hierarchy Theorems for Nonuniform Completeness

We proved unconditionally that every $\leq_{\text{m}}^{\text{P}/\log}$-complete set in NP is $\leq_{\text{T}}^{\text{P}}$-complete. On the other hand, we showed that under the NP-machine hypothesis there exists a $\leq_{\text{m}}^{\text{P}/\text{poly}}$-complete

set in NP that is not $\leq_{\mathrm{T}}^{\mathrm{P}}$-complete. This results in a separation of $\leq_{\mathrm{m}}^{\mathrm{P/poly}}$-completeness from $\leq_{\mathrm{m}}^{\mathrm{P/log}}$-completeness under the NP-machine hypothesis.

▶ **Theorem 11.** *If the* NP*-machine hypothesis is true, then there exists a set in* NP *that is* $\leq_{\mathrm{m}}^{\mathrm{P/poly}}$*-complete, but is not* $\leq_{\mathrm{T}}^{\mathrm{P/log}}$*-complete.*

Proof. Assume the NP-machine hypothesis. From Theorem 2, we obtain a set that is $\leq_{\mathrm{m}}^{\mathrm{P/poly}}$-complete but not $\leq_{\mathrm{T}}^{\mathrm{P}}$-complete. By Theorem 8, this set cannot be $\leq_{\mathrm{T}}^{\mathrm{P/log}}$-complete. ◀

We have the following corollary because the measure hypothesis implies the NP-machine hypothesis.

▶ **Corollary 12.** *If* $\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0$*, then there exists a set in* NP *that is* $\leq_{\mathrm{m}}^{\mathrm{P/poly}}$*-complete, but is not* $\leq_{\mathrm{T}}^{\mathrm{P/log}}$*-complete.*

We note that while Theorem 11 is stated for many-one vs. Turing, it applies to any reducibility in between.

▶ **Corollary 13.** *If the* NP*-machine hypothesis is true, then for any reducibility* $\mathcal{R}$ *where* $\leq_{\mathrm{m}}^{\mathrm{P}}$*-reducibility implies* $\mathcal{R}$*-reducility and* $\mathcal{R}$*-reducility implies* $\leq_{\mathrm{T}}^{\mathrm{P}}$*-reducibility, there is a set in* NP *that is* $\leq_{\mathcal{R}}^{\mathrm{P/poly}}$*-complete, but is not* $\leq_{\mathcal{R}}^{\mathrm{P/log}}$*-complete.*

It is natural to ask if we can separate completeness notions above P/poly many-one. We observe that for this, we will need stronger hypotheses than we have considered in this paper.

▶ **Proposition 14.** *If there is a* $\leq_{\mathrm{T}}^{\mathrm{P/poly}}$*-complete set that is not* $\leq_{\mathrm{m}}^{\mathrm{P/poly}}$*-complete in* NP*, then* NP $\not\subseteq$ P/poly*.*

Proof. If NP $\subseteq$ P/poly, then every set in NP is $\leq_{\mathrm{m}}^{\mathrm{P/poly}}$-complete. ◀

The measure hypothesis and the NP-machine hypothesis are not known to imply NP $\not\subseteq$ P/poly. If it is possible to separate completeness notions above $\leq_{\mathrm{m}}^{\mathrm{P/poly}}$, it appears an additional hypothesis at least as strong as NP $\not\subseteq$ P/poly – such as PH is infinite – would be needed.

References

1. L. Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th IEEE Symposium on Foundations of Computer Science*, pages 75–83, 1978.
2. M. Agrawal. Pseudo-random generators and structure of complete degrees. In *Proceedings of the Seventeenth Annual IEEE Conference on Computational Complexity*, pages 139–147. IEEE Computer Society, 2002.
3. Manindra Agrawal and Osamu Watanabe. One-way functions and the berman-hartmanis conjecture. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*, pages 194–202. IEEE Computer Society, 2009. doi:10.1109/CCC.2009.17.
4. E. Allender. The complexity of complexity. In *Computability and Complexity - Essays Dedicated to Rodney G. Downey on the Occasion of His 60th Birthday*, volume 10010 of *Lecture Notes in Computer Science*, pages 79–94, 2017.
5. E. Allender, H. Buhrman, M. Koucký, D. van Melkebeek, and D. Ronneburger. Power from random strings. *sicomp*, 35:1467–1493, 2006.
6. E. Allender and M. Strauss. Measure on small complexity classes with applications for BPP. In *Proceedings of the 35th Symposium on Foundations of Computer Science*, pages 807–818. IEEE Computer Society, 1994.

7 K. Ambos-Spies and E. Mayordomo. Resource-bounded measure and randomness. In A. Sorbi, editor, *Complexity, Logic and Recursion Theory*, Lecture Notes in Pure and Applied Mathematics, pages 1–47. Marcel Dekker, New York, N.Y., 1997.

8 K. Ambos-Spies, S. A. Terwijn, and X. Zheng. Resource bounded randomness and weakly complete problems. *tcs*, 172(1–2):195–207, 1997.

9 L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

10 J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* Springer-Verlag, Berlin, second edition, 1995.

11 L. Berman and J. Hartmanis. On isomorphism and density of NP and other complete sets. *SICOMP*, 6(2):305–322, 1977.

12 H. Buhrman and D. van Melkebeek. Hard sets are hard to find. *jcss*, 59(2):327–345, 1999. URL: `http://pages.cs.wisc.edu/~dieter/Research/m-complete.html`.

13 Harry Buhrman, Benjamin J. Hescott, Steven Homer, and Leen Torenvliet. Non-uniform reductions. *Theory Comput. Syst.*, 47(2):317–341, 2010. `doi:10.1007/s00224-008-9163-5`.

14 S. Hirahara. Identifying an honest $\mathrm{EXP}^{\mathrm{NP}}$ oracle among many. In *Proceedings of the 30th Conference on Computational Complexity (CCC 2015)*, pages 244–263, 2015.

15 J. M. Hitchcock. The size of SPP. *tcs*, 320(2–3):495–503, 2004. URL: `http://www.cs.uwyo.edu/~jhitchco/papers/sspp.shtml`.

16 J. M. Hitchcock and A. Pavan. Comparing reductions to NP-complete sets. *Information and Computation*, 205(5):694–706, 2007. URL: `http://www.cs.uwyo.edu/~jhitchco/papers/crnpcs.shtml`.

17 J. M. Hitchcock and A. Pavan. Hardness hypotheses, derandomization, and circuit complexity. *Computational Complexity*, 17(1):119–146, 2008. URL: `http://www.cs.uwyo.edu/~jhitchco/papers/hhdcc.shtml`.

18 J. M. Hitchcock and H. Shafei. Autoreducibility of NP-complete sets. In *Proceedings of the 33rd International Symposium on Theoretical Aspects of Computer Science*, pages 42:1–42:12. Leibniz International Proceedings in Informatics, 2016.

19 D. W. Juedes and J. H. Lutz. Weak completeness in E and E_2. *tcs*, 143(1):149–158, 1995.

20 V. Kabanets and J. Y. Cai. Circuit minimization problem. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 73–79, 2000.

21 R. M. Karp and R. J. Lipton. Turing machines that take advice. *Enseign. Math.*, 28:191–201, 1982.

22 A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SICOMP*, 31(5):1501–1526, 2002.

23 J. H. Lutz. Almost everywhere high nonuniform complexity. *jcss*, 44(2):220–258, 1992.

24 J. H. Lutz. Observations on measure and lowness for Δ_2^{P}. *Theory of Computing Systems*, 30(4):429–442, 1997.

25 J. H. Lutz. The quantitative structure of exponential time. In L. A. Hemaspaandra and A. L. Selman, editors, *Complexity Theory Retrospective II*, pages 225–254. Springer-Verlag, 1997.

26 J. H. Lutz and E. Mayordomo. Cook versus Karp-Levin: Separating completeness notions if NP is not small. *tcs*, 164(1–2):141–163, 1996.

27 C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

28 A. Pavan. Comparison of reductions and completeness notions. *SIGACT News*, 34(2):27–41, June 2003. URL: `http://www.cs.iastate.edu/~pavan/papers/sigact.html`.

29 A. Pavan and A. L. Selman. Bi-immunity separates strong NP-completeness notions. *Information and Computation*, 188(1):116–126, 2004.

30 D. Ronneberger. *Kolmogorov Complexity and Derandomization*. PhD thesis, Rutgers University, 2004.

31 C. E. Shannon. The synthesis of two-terminal switching circuits. *Bell System Technical Journal*, 28(1):59–98, 1949.

32 L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *tcs*, 47(3):85–93, 1986.

On the Power of Tree-Depth for Fully Polynomial FPT Algorithms

Yoichi Iwata
National Institute of Informatics, Tokyo, Japan
yiwata@nii.ac.jp

Tomoaki Ogasawara
The University of Tokyo, Tokyo, Japan
t.ogasawara@is.s.u-tokyo.ac.jp

Naoto Ohsaka
The University of Tokyo, Tokyo, Japan
ohsaka@is.s.u-tokyo.ac.jp

Abstract

There are many classical problems in P whose time complexities have not been improved over the past decades. Recent studies of "Hardness in P" have revealed that, for several of such problems, the current fastest algorithm is the best possible under some complexity assumptions. To bypass this difficulty, the concept of "FPT inside P" has been introduced. For a problem with the current best time complexity $O(n^c)$, the goal is to design an algorithm running in $k^{O(1)}n^{c'}$ time for a parameter k and a constant $c' < c$.

In this paper, we investigate the complexity of graph problems in P parameterized by *tree-depth*, a graph parameter related to tree-width. We show that a simple divide-and-conquer method can solve many graph problems, including Weighted Matching, Negative Cycle Detection, Minimum Weight Cycle, Replacement Paths, and 2-hop Cover, in $O(\mathrm{td}\cdot m)$ time or $O(\mathrm{td}\cdot(m + n\log n))$ time, where td is the tree-depth of the input graph. Because any graph of tree-width tw has tree-depth at most $(\mathrm{tw}+1)\log_2 n$, our algorithms also run in $O(\mathrm{tw}\cdot m\log n)$ time or $O(\mathrm{tw}\cdot(m + n\log n)\log n)$ time. These results match or improve the previous best algorithms parameterized by tree-width. Especially, we solve an open problem of fully polynomial FPT algorithm for Weighted Matching parameterized by tree-width posed by Fomin et al. (SODA 2017).

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Fully Polynomial FPT Algorithm, Tree-Depth, Divide-and-Conquer

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.41

Funding Yoichi Iwata was supported by JSPS KAKENHI Grant Number JP17K12643. Naoto Ohsaka was supported by JSPS KAKENHI Grant Number JP16J09440.

Acknowledgements We would like to thank Hiroshi Imai for pointing out to us the reference [21].

1 Introduction

There are many classical problems in P whose time complexities have not been improved over the past decades. For some of such problems, recent studies of "Hardness in P" have provided evidence of why obtaining faster algorithms is difficult. For instance, Vassilevska Williams and Williams [33] and Abboud, Grandoni and Vassilevska Williams [1] showed that many problems including Minimum Weight Cycle, Replacement Paths, and Radius

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 41; pp. 41:1–41:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

are equivalent to All Pair Shortest Paths (APSP) under subcubic reductions; that is, if one of them admits a subcubic-time algorithm, then all of them do.

One of the approaches to bypass this difficulty is to analyze the running time by introducing another measure, called a *parameter*, in addition to the input size. In the theory of parameterized complexity, a problem with a parameter k is called *fixed parameter tractable (FPT)* if it can be solved in $f(k) \cdot |I|^{O(1)}$ time for some function $f(k)$ that does not depend on the input size $|I|$. While the main aim of this theory is to provide fine-grained analysis of NP-hard problems, it is also useful for problems in P. For instance, a simple dynamic programming can solve Maximum Matching in $O(3^{\mathrm{tw}}m)$ time, where m is the number of edges and tw is a famous graph parameter called *tree-width* which intuitively measures how much a graph looks like a tree (see Section 2 for the definition). Therefore, it runs in linear time for any graph of constant tree-width, which is faster than the current best $O(\sqrt{n}m)$ time for the general case [5, 31, 15].

When working on NP-hard problems, we can only expect superpolynomial (or usually exponential) function $f(k)$ in the running time of FPT algorithms. On the other hand, for problems in P, $k^{O(1)}|I|^{O(1)}$-time FPT algorithms might be possible. Such algorithms are called *(fully) polynomial FPT algorithms*, introduced by Giannopoulou, Mertzios and Niedermeier [16]. For instance, Fomin, Lokshtanov, Pilipczuk, Saurabh and Wrochna [11] obtained an $O(\mathrm{tw}^4 \cdot n \log^2 n)$-time (randomized) algorithm for Maximum Matching. In contrast to the $O(3^{\mathrm{tw}}m)$-time dynamic programming, this algorithm is faster than the current best general-case algorithm already for graphs of $\mathrm{tw} = O(n^{\frac{1}{8}-\epsilon})$. In general, for a problem with the current best time complexity $O(n^c)$, the goal is to design an algorithm running in $O(k^d n^{c'})$ time for some small constants d and $c' < c$. Such an algorithm is faster than the current best general-case algorithm already for inputs of $k = O(n^{(c-c')/d-\epsilon})$. On the negative side, Abboud, Vassilevska Williams and Wang [2] showed that Diameter and Radius do not admit $2^{o(\mathrm{tw})}n^{2-\epsilon}$-time algorithms under some plausible assumptions. In this paper, we give new or improved fully polynomial FPT algorithms for several classical graph problems. Especially, we solve an open problem for Weighted Matching posed by Fomin et al. [11].

Our approach. Before describing our results, we first give a short review of existing work on fully polynomial FPT algorithms parameterized by tree-width and explain our approach. There are roughly three types of approaches in the literature. The first approach is to use a polynomial-time dynamic programming on a tree-decomposition, which has been mainly used for problems related to shortest paths [7, 27, 4, 32]. The second approach is to use an $O(\mathrm{tw}^3 \cdot n)$-time Gaussian elimination of matrices of small tree-width developed by Fomin et al. [11]. The above-mentioned $O(\mathrm{tw}^4 \cdot n \log^2 n)$-time algorithm for Maximum Matching was obtained by this approach. The third approach is to apply a divide-and-conquer method exploiting the existence of small *balanced separators*. This approach was first used for planar graphs by Lipton and Tarjan [21]. Using the existence of $O(\sqrt{n})$-size balanced separators, they obtained an $O(n^{1.5})$-time algorithm for Maximum Matching and an $O(n^{1.5} \log n)$-time algorithm for Weighted Matching for planar graphs. For graphs of bounded tree-width, Akiba, Iwata and Yoshida [3] obtained an $O(\mathrm{tw} \cdot (m + n \log n) \log n)$-time algorithm for 2-hop Cover, which is a problem of constructing a distance oracle, and Fomin et al. [11] obtained an $O(\mathrm{tw} \cdot m \log n)$-time[1] algorithm for Vertex-disjoint $s-t$

[1] While the running time shown in [11] is $O(\mathrm{tw}^2 \cdot n \log n)$, we can easily see that it also runs in $O(\mathrm{tw} \cdot m \log n)$ time. Because $m = O(\mathrm{tw} \cdot n)$ holds for any graphs of tree-width tw, the latter is never worse than the former. Note that $\mathrm{tw} \cdot n$ in the running time of other algorithms cannot be replaced by m in general;

Table 1 Comparison of previous results and our results. n and m denote the number of vertices and edges, w denotes the width of the given tree-decomposition, and d denotes the depth of the given elimination forest. The factor d in our results can be replaced by $w \cdot \log n$.

Problem	Previous result	Our result
Maximum Matching	$O(w^4 n \log^2 n)$ [11]	$O(dm)$
Weighted Matching	Open problem [11]	$O(d(m + n \log n))$
Negative Cycle Detection	$O(w^2 n)$ [27]	$O(d(m + n \log n))$
Minimum Weight Cycle	—	$O(d(m + n \log n))$
Replacement Paths	—	$O(d(m + n \log n))$
2-hop Cover	$O(w(m + n \log n) \log n)$ [3]	$O(d(m + n \log n))$

Paths. We obtain fully polynomial FPT algorithms for a wide range of problems by using this approach. Our key observation is that, when using the divide-and-conquer approach, another graph parameter called *tree-depth* is more powerful than the tree-width.

A graph G of tree-width tw admits a set S of $\mathrm{tw} + 1$ vertices, called a balanced separator, such that each connected component of $G - S$ contains at most $\frac{n}{2}$ vertices. In both of the above-mentioned divide-and-conquer algorithms for graphs of bounded tree-width, after the algorithm recursively computes a solution for each connected component of $G - S$, it constructs a solution for G in $O(\mathrm{tw} \cdot (m + n \log n))$ time or $O(\mathrm{tw} \cdot m)$ time, respectively. Because the depth of the recursive calls is bounded by $O(\log n)$, the total running time becomes $O(\mathrm{tw} \cdot (m + n \log n) \log n)$ or $O(\mathrm{tw} \cdot m \log n)$, respectively.

Here, we observe that, by using tree-depth, this kind of divide-and-conquer algorithm can be simplified and the analysis can be improved. Tree-depth is a graph parameter which has been studied under various names [29, 20, 6, 25]. A graph has tree-depth td if and only if there exists an *elimination forest* of depth td. See Section 2 for the precise definition of the tree-depth and the elimination forest. An important property of tree-depth is that any connected graph G of tree-depth td can be divided into connected components of tree-depth at most $\mathrm{td} - 1$ by removing a single vertex r. Therefore, if there exists an $O(m)$-time or $O(m + n \log n)$-time *incremental algorithm*, which constructs a solution for G from a solution for $G - r$, we can solve the problem in $O(\mathrm{td} \cdot m)$ time or $O(\mathrm{td} \cdot (m + n \log n))$ time, respectively. Now, the only thing to do is to develop such an incremental algorithm for each problem. We present a detailed discussion of this framework in Section 3. Because any graph of tree-width tw has tree-depth at most $(\mathrm{tw} + 1) \log_2 n$ [24], the running time can also be bounded by $O(\mathrm{tw} \cdot m \log n)$ or $O(\mathrm{tw} \cdot (m + n \log n) \log n)$. Therefore, our analysis using tree-depth is never worse than the existing results directly using tree-width. On the other hand, there are infinitely many graphs whose tree-depth has asymptotically the same bound as tree-width. For instance, if every N-vertex subgraph admits a balanced separator of size $O(N^\alpha)$ for some constant $\alpha > 0$ (e.g., $\alpha = \frac{1}{2}$ for H-minor free graphs), both tree-width and tree-depth are $O(n^\alpha)$. Hence, for such graphs, the time complexity using tree-depth is truly better than that using tree-width.

Our results. Table 1 shows our results and the comparison to the existing results on fully polynomial FPT algorithms parameterized by tree-width. The formal definition of each problem is given in Section 4. Because obtaining an elimination forest of the lowest depth

e.g., we cannot bound the running time of the Gaussian elimination by $O(\mathrm{tw}^2 \cdot m)$, where m is the number of non-zero elements.

is NP-hard, we assume that an elimination forest is given as an input and the parameter for our results is the depth d of the given elimination forest. Similarly, for the existing results, the parameter is the width w of the given tree-decomposition. Note that, because a tree-decomposition of width w can be converted into an elimination forest of depth $O(w \cdot \log n)$ in linear time [29], we can always replace the factor d in our running time by $w \cdot \log n$. This also means that we can use arbitrary approximation algorithms or heuristics for constructing tree-decompositions for obtaining an elimination forest.

The first polynomial-time algorithms for Maximum Matching and Weighted Matching were obtained by Edmonds [10], and the current fastest algorithms run in $O(\sqrt{n}m)$ time [5, 31, 15] and $O(n(m + n \log n))$ time [5], respectively. Fomin et al. [11] obtained the $O(w^4 n \log^2 n)$-time randomized algorithm for Maximum Matching by using an algebraic method and the fast computation of Gaussian elimination. They left as an open problem whether a similar running time is possible for Weighted Matching. The general-case algorithms for these problems compute a maximum matching by iteratively finding an *augmenting path*, and therefore, they are already incremental. Thus, we can easily obtain an $O(dm)$-time algorithm for Maximum Matching and an $O(d(m + n \log n))$-time algorithm for Weighted Matching. Note that the divide-and-conquer algorithms for planar matching by Lipton and Tarjan [21] also use this augmenting-path approach, and our result can be seen as extension to bounded tree-depth graphs. Our algorithm for Maximum Matching is always faster than the one by Fomin et al. because we have $m = O(kn)$ for any graph of tree-width or tree-depth k and is faster than the general-case algorithm already when $d = O(n^{\frac{1}{2}-\epsilon})$. Our algorithm for Weighted Matching settles the open problem and is faster than the general-case algorithm already when $d = O(n^{1-\epsilon})$.

The current fastest algorithm for Negative Cycle Detection is the classical $O(nm)$-time Bellman-Ford algorithm. Planken et al. [27] obtained an $O(w^2 n)$-time algorithm by using a Floyd-Warshall-like dynamic programming. In this paper, we give an $O(d(m+n \log n))$-time algorithm. While the algorithm by Planken et al. is faster than the general-case algorithm only when $w = O(m^{\frac{1}{2}-\epsilon})$, our algorithm achieves a faster running time already when $d = O(n^{1-\epsilon})$.

Both Minimum Weight Cycle (or Girth) and Replacement Paths are subcubic-equivalent to APSP [33]. A naive algorithm can solve both problems in $O(n^3)$ time or $O(n(m+ n \log n))$ time. For Minimum Weight Cycle of directed graphs, an improved $O(nm)$-time algorithm was recently obtained by Orlin and Sedeño-Noda [26]. For Replacement Paths, Malik et al. [22] obtained an $O(m+n \log n)$-time algorithm for undirected graphs, and Roditty and Zwick [28] obtained an $O(\sqrt{n}m \cdot \text{polylog}\, n)$-time algorithm for unweighted graphs. For the general case, Gotthilf and Lewenstein [17] obtained an $O(n(m + n \log \log n))$-time algorithm, and there exists an $\Omega(\sqrt{n}m)$-time lower bound in the path-comparison model [19] (whenever $m = O(n\sqrt{n})$) [18]. In this paper, we give an $O(d(m + n \log n))$-time algorithm for each of these problems, which is faster than the general-case algorithm already when $d = O(n^{1-\epsilon})$. This result shows the following contrast to the known result of "Hardness in P": Radius is also subcubic-equivalent to APSP [1] but it cannot be solved in a similar running time under some plausible assumptions [2].

2-hop cover [8] is a data structure for answering distance queries in an efficient manner. Akiba et al. [3] obtained an $O(w(m + n \log n) \log n)$-time algorithm for constructing a 2-hop cover answering each distance query in $O(w \log n)$ time. In this paper, we give an $O(d(m + n \log n))$-time algorithm for constructing a 2-hop cover answering each distance query in $O(d)$ time.

Related work. Coudert, Ducoffe and Popa [9] have developed fully polynomial FPT algorithms using several other graph parameters including clique-width. In contrast to the tree-depth, their parameters are not polynomially bounded by tree-width, and therefore, their results do not imply fully polynomial FPT algorithms parameterized by tree-width. Mertzios, Nichterlein and Niedermeier [23] have obtained an $O(m + k^{1.5})$-time algorithm for Maximum Matching parameterized by feedback edge number k ($= m - n + 1$ when the graph is connected) by giving a linear-time kernel.

2 Preliminaries

Let $G = (V, E)$ be a directed or undirected graph, where V is a set of vertices of G and E is a set of edges of G. When the graph is clear from the context, we use n to denote the number of vertices and m to denote the number of edges. All the graphs in this paper are simple (i.e., they have no multiple edges nor self-loops). Let $S \subseteq V$ be a subset of vertices. We denote by $E[S]$ the set of edges whose endpoints are both in S and denote by $G[S]$ the subgraph induced by S (i.e., $G[S] = (S, E[S])$).

A *tree decomposition* of a graph $G = (V, E)$ is a pair (T, B) of a tree $T = (X, F)$ and a collection of *bags* $\{B_x \subseteq V \mid x \in X\}$ satisfying the following two conditions.

- For each edge $uv \in E$, there exists some $x \in X$ such that $\{u, v\} \subseteq B_x$.
- For each vertex $v \in V$, the set $\{x \in X \mid v \in B_x\}$ induces a connected subtree in T.

The *width* of (T, B) is the maximum of $|B_x| - 1$ and the *tree-width* $\mathrm{tw}(G)$ of G is the minimum width among all possible tree decompositions.

An *elimination forest* T of a graph $G = (V, E)$ is a rooted forest on the same vertex set V such that, for every edge $uv \in E$, one of u and v is an ancestor of the other. The *depth* of T is the maximum number of vertices on a path from a root to a leaf in T. The *tree-depth* $\mathrm{td}(G)$ of a graph G is the minimum depth among all possible elimination forests. Tree-width and tree-depth are strongly related as the following lemma shows.

▶ **Lemma 1** ([24, 29]). *For any graph G, the following holds.*

$$\mathrm{tw}(G) + 1 \leq \mathrm{td}(G) \leq (\mathrm{tw}(G) + 1) \log_2 n.$$

Moreover, given a tree decomposition of width k, we can construct an elimination forest of depth $O(k \log n)$ in linear time.

3 Divide-and-conquer framework

In this section, we propose a divide-and-conquer framework that can be applicable to a wide range of problems parameterized by tree-depth.

▶ **Theorem 2.** *Let $G = (V, E)$ be a graph and let f be a function defined on subsets of V. Suppose that $f(\emptyset)$ can be computed in constant time and we have the following two algorithms* Increment *and* Union *with time complexity $T(n, m)(= \Omega(n + m))$.*

- Increment$(X, f(X), x) \mapsto f(X \cup \{x\})$. *Given a set $X \subseteq V$, its value $f(X)$, and a vertex $x \notin X$, this algorithm computes the value $f(X \cup \{x\})$ in $T(|X \cup \{x\}|, |E[X \cup \{x\}]|)$ time.*
- Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c))) \mapsto f(\bigcup_i X_i)$. *Given disjoint sets $X_1, \ldots, X_c \subseteq V$ such that G has no edges between X_i and X_j for any $i \neq j$, and their values $f(X_1), \ldots, f(X_c)$, this algorithm computes the value $f(\bigcup_i X_i)$ in $T(|\bigcup_i X_i|, |E[\bigcup_i X]|)$ time.*

Then, for a given elimination forest of G of depth k, we can compute the value $f(V)$ in $O(k \cdot T(n, m))$ time.

Algorithm 1 Algorithm for computing $f(V)$.

```
1: procedure Compute(S, T_S) ↦ f(S)        ▷ T_S is an elimination forest of G[S].
2:     if S = ∅ then return f(∅)
3:     T_1, ..., T_c ← the connected trees of T_S
4:     X_1, ..., X_c ← the sets of vertices of T_1, ..., T_c
5:     for i ∈ {1, ..., c} do
6:         x_i ← the root of T_i
7:         f_i ← Increment(X_i \ {x_i}, Compute(X_i \ {x_i}, T_i − x_i), x_i)
8:     return Union((X_1, f_1), ..., (X_c, f_c))
```

Proof. Algorithm 1 describes our divide-and-conquer algorithm. We prove that for any set S and any elimination forest T_S of $G[S]$ of depth k_S, $\textsc{Compute}(S, T_S)$ correctly computes the value $f(S)$ in $(2k_S + 1) \cdot T(|S|, |E[S]|)$ time by induction on the size of S.

The claim trivially holds when $S = \emptyset$. For a set $S \neq \emptyset$, let $T_1, \ldots, T_c$ be the connected trees of T_S ($c = 1$ if T_S is connected). For each i, let X_i be the set of vertices of T_i. From the definition of the elimination forest, G has no edges between X_i and X_j for any $i \neq j$. For each i, we compute the value $f(X_i)$ as follows. Let x_i be the root of T_i. By removing x_i from T_i, we obtain an elimination forest of $G[X_i \setminus \{x_i\}]$ of depth at most $k_S - 1$. Therefore, by the induction hypothesis, we can correctly compute the value $f(X_i \setminus \{x_i\})$ in $(2k_S - 1) \cdot T(|X_i|, |E[X_i]|)$ time. Then, by applying $\textsc{Increment}(X_i \setminus \{x_i\}, f(X_i \setminus \{x_i\}), x_i)$, we obtain the value $f(X_i)$ in $2k_S \cdot T(|X_i|, |E[X_i]|)$ time. Because $|S| = \sum_i |X_i|$ and $|E[S]| = \sum_i |E[X_i]|$ hold, the total running time of these computations is $2k_S \cdot \sum_i T(|X_i|, |E[X_i]|) \leq 2k_S \cdot T(|S|, |E[S]|)$. Finally, by applying the algorithm Union, we obtain the value $f(S)$ in $(2k_S + 1) \cdot T(|S|, |E[S]|)$ time. ◀

Note that the algorithm Union is trivial in most applications. We have only one non-trivial case in Section 4.5 in this paper. From the relation between tree-depth and tree-width (Lemma 1), we obtain the following corollary.

▶ **Corollary 3.** *Under the same assumption as in Theorem 2, for a given tree decomposition of G of width k, we can compute the value $f(V)$ in $O(k \cdot T(n, m) \log n)$ time.*

4 Applications

4.1 Maximum matching

For an undirected graph $G = (V, E)$, a *matching* M of G is a subset of E such that no edges in M share a vertex. In this section, we prove the following theorem.

▶ **Theorem 4.** *Given an undirected graph and its elimination forest of depth k, we can compute a maximum-size matching in $O(km)$ time.*

As mentioned in the introduction, we use the augmenting-path approach, which is also used for planar matching [21]. Let M be a matching. A vertex not incident to M is called *exposed*. An *M-alternating path* is a (simple) path whose edges are alternately out of and in M. An M-alternating path connecting two different exposed vertices is called an *M-augmenting path*. If there exists an M-augmenting path P, by taking the symmetric difference $M \Delta E(P)$, where $E(P)$ is the set of edges in P, we can construct a matching of size $|M| + 1$. In fact, M is the maximum-size matching if and only if there exist no M-augmenting paths. Edmonds [10] developed the first polynomial-time algorithm for computing an M-augmenting path by introducing the notion of blossom, and an $O(m)$-time algorithm was given by Gabow and Tarjan [14].

▶ **Lemma 5** ([14]). *Given an undirected graph and its matching M, we can either compute a matching of size $|M|+1$ or correctly conclude that M is a maximum-size matching in $O(m)$ time.*

For $S \subseteq V$, we define $f(S)$ as a function that returns a maximum-size matching of $G[S]$. We now give INCREMENT and UNION.

Increment$(X, f(X), x)$. Because the size of the maximum matching of $G[X \cup \{x\}]$ is at most the size of the maximum matching of $G[X]$ plus one, we can compute a maximum matching of $G[X \cup \{x\}]$ in $O(|E[X \cup \{x\}]|)$ time by a single application of Lemma 5.

Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c)))$. Because there exist no edges between X_i and X_j for any $i \neq j$, we can construct a maximum matching of $G[\bigcup_i X_i]$ just by taking the union of $f(X_i)$.

Proof of Theorem 4. The algorithm INCREMENT$(X, f(X), x)$ correctly computes $f(X \cup \{x\})$ in $O(|E[X \cup \{x\}]|)$ time and the algorithm UNION$((X_1, f(X_1)), \ldots, (X_c, f(X_c)))$ correctly computes $f(\bigcup_i X_i)$ in $O(|\bigcup_i X_i|)$ time. Therefore, from Theorem 2, we can compute a maximum-size matching of G in $O(km)$ time. ◀

4.2 Weighted matching

Let $G = (V, E)$ be an undirected graph with an edge-weight function $w : E \to \mathbb{R}$. A weight of a matching M, denoted by $w(M)$, is simply defined as the total weight of edges in M. A matching M of G is called *perfect* if G has no exposed vertices (or equivalently $|M| = \frac{n}{2}$). A perfect matching is called a *maximum-weight perfect matching* if it has the maximum weight among all perfect matchings of G. We can easily see that other variants of weighted matching problems can be reduced to the problem of finding a maximum-weight perfect matching even when parameterized by tree-depth. In this section, we prove the following theorem.

▶ **Theorem 6.** *Given an edge-weighted undirected graph admitting at least one perfect matching and its elimination forest of depth k, we can compute a maximum-weight perfect matching in $O(k(m + n \log n))$ time.*

In our algorithm, we use an $O(n(m + n \log n))$-time primal-dual algorithm by Gabow [12]. In this primal-dual algorithm, we keep a pair of a matching M and dual variables (Ω, y, z), where Ω is a laminar[2] collection of odd-size subsets of V and y and z are functions $y : V \to \mathbb{R}$ and $z : \Omega \to \mathbb{R}_{\geq 0}$, satisfying the following conditions:

$$\widehat{yz}(uv) := y(u) + y(v) + \sum_{B \in \Omega : u, v \in B} z(B) \geq w(uv) \qquad \text{for every } uv \in E, \tag{1}$$

$$\widehat{yz}(uv) = w(uv) \qquad \text{for every } uv \in M, \tag{2}$$

$$|\{uv \in M \mid u, v \in B\}| = \left\lfloor \frac{|B|}{2} \right\rfloor \qquad \text{for every } B \in \Omega. \tag{3}$$

From the duality theory (see e.g. [13]), a perfect matching M is a maximum-weight perfect matching if and only if there exist dual variables (Ω, y, z) satisfying the above conditions. Gabow [12] obtained the $O(n(m + n \log n))$-time algorithm by iteratively applying the following lemma.

[2] A collection Ω of subsets of a ground set V is called *laminar* if for any $X, Y \in \Omega$, one of $X \cap Y = \emptyset$, $X \subseteq Y$, or $X \subseteq Y$ holds. When Ω is laminar, we have $|\Omega| = O(|V|)$.

▶ **Lemma 7** ([12]). *Given an edge-weighted undirected graph and a pair of a matching M and dual variables (Ω, y, z) satisfying the conditions (1)–(3), we can either compute a pair of a matching M' of cardinality $|M|+1$ and dual variables (Ω', y', z') satisfying the conditions (1)–(3) or correctly conclude that M is a maximum-size matching*[3] *in $O(m + n \log n)$ time.*

For $S \subseteq V$, we define $f(S)$ as a function that returns a pair of a maximum-size matching M_S of $G[S]$ and dual variables (Ω_S, y_S, z_S) satisfying the conditions (1)–(3). We now give Increment and Union.

Increment$(X, f(X), x)$. Let W be a value satisfying $W + y_X(v) \geq w(xv)$ for every $xv \in E[X \cup \{x\}]$. Let $y : X \cup \{x\} \to \mathbb{R}$ be a function defined as $y(x) := W$ and $y(v) := y_X(v)$ for $v \in X$. In the subgraph $G[X \cup \{x\}]$, a pair of the matching M_X and dual variables (Ω_X, y, z_X) satisfies the conditions (1)–(3). Therefore, we can apply Lemma 7. If M_X is a maximum-size matching of $G[X \cup \{x\}]$, we return M_X and (Ω_X, y, z_X). Otherwise, we obtain a matching M' of size $|M_X|+1$ and dual variables (Ω', y', z') satisfying the conditions (1)–(3). Because the cardinality of maximum-size matching of $G[X \cup \{x\}]$ is at most the cardinality of maximum-size matching of $G[X]$ plus one, the obtained M' is a maximum-size matching of $G[X \cup \{x\}]$. Therefore, we can return M' and (Ω', y', z').

Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c)))$. Because there exist no edges between X_i and X_j for any $i \neq j$, we can simply return a pair of a maximum-size matching obtained by taking the union $\bigcup_i M_{X_i}$ and dual variables (Ω, y, z) such that $\Omega := \bigcup_i \Omega_{X_i}$, $y(v) := y_{X_i}(v)$ for $v \in X_i$, and $z(B) = z_{X_i}(B)$ for $B \in \Omega_{X_i}$.

Proof of Theorem 6. The algorithm Increment$(X, f(X), x)$ runs in $O(|E[X \cup \{x\}]| + |X| \log |X|)$ time and the algorithm Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c)))$ runs in $O(|\bigcup X_i|)$ time. Therefore, from Theorem 2, we can compute $f(V)$ in $O(k(m + n \log n))$ time. From the duality theory, the perfect matching obtained by computing $f(V)$ is a maximum-weight perfect matching of G. ◀

4.3 Negative cycle detection and potentials

Let $G = (V, E)$ be a directed graph with an edge-weight function $w : E \to \mathbb{R}$. For a function $p : V \to \mathbb{R}$, we define an edge-weight function w_p as $w_p(uv) := w(uv) + p(u) - p(v)$. If w_p becomes non-negative for all edges, p is called a *potential on G*.

▶ **Lemma 8** ([30]). *There exists a potential on G if and only if G has no negative cycles.*

In this section, we prove the following theorem.

▶ **Theorem 9.** *Given an edge-weighted directed graph and its elimination forest of depth k, we can compute either a potential or a negative cycle in $O(k(m + n \log n))$ time.*

Suppose that we have a potential p. Because w_p is non-negative, we can compute a shortest-path tree rooted at a given vertex s under w_p in $O(m + n \log n)$ time with Dijkstra's algorithm. For any $s - t$ path, its length under w_p is exactly the length under w plus a constant $p(s) - p(t)$. Therefore, the obtained tree is also a shortest-path tree under w. Thus, we obtain the following corollary.

[3] Note that when M is not a perfect matching, this does not imply that M has the maximum weight among all the maximum-size matchings.

▶ **Corollary 10.** *Given an edge-weighted directed graph without negative cycles, a vertex s, and its elimination forest of depth k, we can compute a shortest-path tree rooted at s in $O(k(m + n \log n))$ time.*

For $S \subseteq V$, we define $f(S)$ as a function that returns either a potential $p_S : S \to \mathbb{R}$ on $G[S]$ or a negative cycle contained in $G[S]$. We now give Increment and Union.

Increment$(X, f(X), x)$. If $f(X)$ is a negative cycle, we return it. Otherwise, let $G' = (X \cup \{x\}, E')$ be the graph obtained from $G[X \cup \{x\}]$ by removing all the edges incoming to x. Let W be a value satisfying $w(xv) + W - p_X(v) \geq 0$ for every $xv \in E'$. Let $p' : X \cup \{x\} \to \mathbb{R}$ be a function defined as $p'(x) := W$ and $p'(v) := p_X(v)$ for $v \in X$. Because x has no incoming edges in G', p' is a potential on G'. Therefore, we can compute a shortest-path tree rooted at x under $w_{p'}$ in $O(|E[X]| + |X| \log |X|)$ time with Dijkstra's algorithm. Let R be the set of vertices reachable from x in G' and let $d : R \to \mathbb{R}$ be the shortest-path distance from x under $w_{p'}$. If there exists an edge $vx \in E[X \cup \{x\}]$ such that $v \in R$ and $d(v) + w_{p'}(vx) < 0$, $G[X \cup \{x\}]$ contains a negative cycle starting from x, going to v along the shortest-path tree, and coming back to x via the edge vx. Otherwise, let D be a value satisfying $w_{p'}(uv) + D - d(v) \geq 0$ for every $uv \in E[X \cup \{x\}]$ with $u \in X \setminus R$ and $v \in R$. Then, we return a function $p : X \cup \{x\} \to \mathbb{R}$ defined as $p(v) := p'(v) + d(v)$ if $v \in R$ and $p(v) := p'(v) + D$ if $v \in X \setminus R$.

▶ **Claim 1.** *p is a potential on $G[X \cup \{x\}]$.*

Proof. For every edge $uv \in E[X \cup \{x\}]$, we have

$$w_p(uv) = \begin{cases} w_{p'}(uv) + d(u) - d(v) \geq 0 & \text{if } u, v \in R, \\ w_{p'}(uv) + D - d(v) \geq 0 & \text{if } u \in X \setminus R, v \in R, \\ w_{p'}(uv) + D - D \geq 0 & \text{if } u \in X \setminus R, v \in X \setminus R. \end{cases}$$

Note that there are no edges from R to $X \setminus R$. ◀

Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c)))$. If at least one of $f(X_i)$ is a negative cycle, we return it. Otherwise, we return a potential p defined as $p(v) := p_{X_i}(v)$ for $v \in X_i$.

Proof of Theorem 9. The algorithm Increment$(X, f(X), x)$ correctly computes $f(X \cup \{x\})$ in $O(|E[X]| + |X| \log |X|)$ time and the algorithm Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c)))$ correctly computes $f(\bigcup_i X_i)$ in $O(|\bigcup_i X_i|)$ time. Thus, from Theorem 2, we can compute $f(V)$, i.e., either a potential on G or a negative cycle contained in G, in $O(k(m + n \log n))$ time. ◀

4.4 Minimum weight cycle

In this section, we prove the following theorem.

▶ **Theorem 11.** *Given a non-negative edge-weighted undirected or directed graph and its elimination forest of depth k, we can compute a minimum-weight cycle in $O(k(m + n \log n))$ time.*

Note that when the graph is undirected, a closed walk of length two using the same edge twice is not considered as a cycle. Therefore, we cannot simply reduce the undirected version into the directed version by replacing each undirected edge by two directed edges of both directions.

Let $G = (V, E)$ be the input graph with an edge-weight function $w : E \to \mathbb{R}_{\geq 0}$. For $S \subseteq V$, we define $f(S)$ as a function that returns a minimum-weight cycle of $G[S]$. We describe INCREMENT and UNION below.

Increment$(X, f(X), x)$. Because we have a minimum-weight cycle $f(X)$ of $G[X]$, we only need to find a minimum-weight cycle passing through x. First, we construct a shortest-path tree of $G[X \cup \{x\}]$ rooted at x and let $d : X \cup \{x\} \to \mathbb{R}$ be the shortest-path distance.

When the graph is undirected, we find an edge $uv \in E[X \cup \{x\}]$ not contained in the shortest-path tree minimizing $d(u) + w(uv) + d(v)$. If this weight is at least the weight of $f(X)$, we return $f(X)$. Otherwise, we return the cycle starting from x, going to u along the shortest-path tree, jumping to v through the edge uv, and coming back to x along the shortest-path tree. Note that this always forms a cycle because otherwise, it induces a cycle contained in $G[X]$ that has a smaller weight than $f(X)$, which is a contradiction.

We can prove the correctness of this algorithm as follows. Let W be the weight of the cycle obtained by the algorithm and let C be a cycle passing through x. Let $v_0 = x, v_1, \ldots, v_{\ell-1}, v_\ell = x$ the vertices on C in order. Because a tree contains no cycles, there exists an edge $v_i v_{i+1}$ not contained in the shortest-path tree. Therefore, the weight of C is $\sum_{j=0}^{i-1} w(v_j v_{j+1}) + w(v_i v_{i+1}) + \sum_{j=i+1}^{\ell-1} w(v_j v_{j+1}) \geq d(v_i) + w(v_i v_{i+1}) + d(v_{i+1}) \geq W$.

When the graph is directed, we find an edge $ux \in E[X \cup \{x\}]$ with the minimum $d(u) + w(ux)$. If this weight is at least the weight of $f(X)$, we return $f(X)$. Otherwise, we return the cycle starting from x, going to u along the shortest-path tree, and coming back to x through the edge ux.

Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c)))$. We return a cycle of the minimum weight among $f(X_1), \ldots, f(X_c)$.

Proof of Theorem 11. The algorithm INCREMENT$(X, f(X), x)$ correctly computes $f(X \cup \{x\})$ in $O(|E[X]| + |X| \log |X|)$ time and the algorithm UNION$((X_1, f(X_1)), \ldots, (X_c, f(X_c))$ correctly computes $f(\bigcup_i X_i)$ in $O(|\bigcup_i X_i|)$ time. Therefore, from Theorem 2, we can compute a minimum-weight cycle in $O(k(m + n \log n))$ time. ◀

4.5 Replacement paths

Fix two vertices s and t. For an edge-weighed directed graph $G = (V, E)$ and an edge $e \in E$, we denote the length of the shortest $s - t$ path avoiding e by $r_G(e)$. In this section, we prove the following theorem.

▶ **Theorem 12.** *Given an edge-weighted directed graph $G = (V, E)$, a shortest $s - t$ path P, and its elimination forest of depth k, we can compute $r_G(e)$ for all edges e on P in $O(k(m + n \log n))$ time.*

Let $v_0(= s), v_1, \ldots, v_{\ell-1}, v_\ell(= t)$ be the vertices on the given shortest $s - t$ path P in order. For $i \in \{0, \ldots, \ell\}$, we denote the length of the prefix $v_0 v_1 \ldots v_i$ by $\mathrm{pref}(v_i)$ and the length of the suffix $v_i v_{i+1} \ldots v_\ell$ by $\mathrm{suf}(v_i)$. These can be precomputed in linear time.

For $S \subseteq V$, we define $G[S] \cup P$ as a graph consisting of vertices $S \cup \{v_0, \ldots, v_\ell\}$ and edges $E[S] \cup \{v_0 v_1, \ldots, v_{\ell-1} v_\ell\}$, and define $G[S] \setminus P$ as a graph consisting of vertices S and edges $E[S] \setminus \{v_0 v_1, \ldots, v_{\ell-1} v_\ell\}$. We denote the shortest-path length from u to v in $G[S] \setminus P$ by $d_S(u, v)$. For convenience, we define $d_S(u, v) = \infty$ when $u \notin S$ or $v \notin S$. We use the following lemma.

▶ **Lemma 13.** *For any $S \subseteq V$ and any $i \in \{0, \dots, \ell - 1\}$, $r_{G[S]\cup P}(v_i v_{i+1})$ is the minimum of $\mathrm{pref}(v_a) + d_S(v_a, v_b) + \mathrm{suf}(v_b)$ for $a \le i < b$.*

Proof. Any $s - t$ path avoiding $v_i v_{i+1}$ in $G[S] \cup P$ can be written as, for some $a \le i < b$, a concatenation of $s - v_a$ path Q_1, $v_a - v_b$ path Q_2 that is contained in $G[S] \setminus P$, and $v_b - t$ path Q_3. Because P is a shortest $s - t$ path in G, we can replace Q_1 by the prefix $v_0 \dots v_a$, Q_2 by the shortest $v_a - v_b$ path in $G[S] \setminus P$, and Q_3 by the suffix $v_b \dots v_\ell$ without increasing the length. Therefore, the lemma holds. ◀

We want to define $f(S)$ as a function that returns a list of $r_{G[S]\cup P}(v_i v_{i+1})$ for all $i \in \{0, \dots, \ell - 1\}$; however, we cannot do so because the length of this list is not bounded by $|S|$. Instead, we define $f(S)$ as a function that returns a list of $r_{G[S]\cup P}(v_i v_{i+1})$ for all i with $v_i \in S$. This succinct representation has enough information because, for any $v_i \notin S$, we have $r_{G[S]\cup P}(v_i v_{i+1}) = r_{G[S]\cup P}(v_{i-1} v_i)$ (or ∞ when $i = 0$). We describe Increment and Union below.

Increment$(X, f(X), x)$. By running Dijkstra's algorithm twice, we compute $d_{X\cup\{x\}}(x, v)$ and $d_{X\cup\{x\}}(v, x)$ for all $v \in X \cup \{x\}$ in $O(|E[X]| + |X| \log |X|)$ time. For $v_i \in X \cup \{x\}$, we define $L_i := \min_{a \le i, v_a \in X\cup\{x\}} (\mathrm{pref}(v_a) + d(v_a, x))$ and $R_i := \min_{b > i, v_b \in X\cup\{x\}} (d(x, v_b) + \mathrm{suf}(v_b))$. By a standard dynamic programming, we can compute L_i and R_i for all i with $v_i \in X \cup \{x\}$ in $O(|X|)$ time.

From Lemma 13, $r_{G[X\cup\{x\}]\cup P}(v_i v_{i+1}) = \mathrm{pref}(v_a) + d_{X\cup\{x\}}(v_a, v_b) + \mathrm{suf}(v_b)$ holds for some $a \le i < b$. If $d_{X\cup\{x\}}(v_a, v_b) = d_X(v_a, v_b)$ holds, we have $r_{G[X\cup\{x\}]\cup P}(v_i v_{i+1}) = r_{G[X]\cup P}(v_i v_{i+1})$, and otherwise, we have $d_{X\cup\{x\}}(v_a, v_b) = d_{X\cup\{x\}}(v_a, x) + d_{X\cup\{x\}}(x, v_b)$. Therefore, we can compute $r_{G[X\cup\{x\}]\cup P}(v_i v_{i+1})$ by taking the minimum of $r_{G[X]\cup P}(v_i v_{i+1})$ and $\min_{a \le i < b} (\mathrm{pref}(v_a) + d(v_a, x) + d(x, v_b) + \mathrm{suf}(v_b)) = L_i + R_i$.

Union$((X_1, f(X_1)), \dots, (X_c, f(X_c)))$. Let $X := \bigcup_i X_i$. Because there exist no edges between X_i and X_j for any $i \ne j$, we have $d_X(u, v) = \min_i d_{X_i}(u, v)$ for any $u, v \in X$. Therefore, from Lemma 13, we have $r_{G[X]\cup P}(v_i v_{i+1}) = \min_j r_{G[X_j]\cup P}(v_i v_{i+1})$. For efficiently computing $r_{G[X]\cup P}(v_i v_{i+1})$ for all i with $v_i \in X$, we do as follows in increasing order of i.

For each X_j, we maintain a value r_j so that $r_j = r_{G[X_j]\cup P}(v_i v_{i+1})$ always holds. Initially, these values are set to ∞. We use a heap for computing $\min_j r_j$ and updating r_j in $O(\log c)$ time. For processing i, we first update $r_j \leftarrow r_{G[X_j]\cup P}(v_i v_{i+1})$ for the set X_j containing v_i. We do not need to update $r_{j'}$ for any other set $X_{j'}$ because $r_{G[X_{j'}]\cup P}(v_i v_{i+1}) = r_{G[X_{j'}]\cup P}(v_{i-1} v_i)$ holds. Then, we compute $r_{G[X]\cup P}(v_i v_{i+1}) = \min_j r_j$.

Proof of Theorem 12. The algorithm Increment$(X, f(X), x)$ correctly computes $f(X \cup \{x\})$ in $O(|E[X]| + |X| \log |X|)$ time and the algorithm Union$((X_1, f(X_1)), \dots, (X_c, f(X_c)))$ correctly computes $f(\bigcup_i X_i)$ in $O(|\bigcup_i X_i| \log c) = O(|\bigcup_i X_i| \log |\bigcup_i X_i|)$ time. Therefore, from Theorem 2, we can compute $f(V)$, i.e., $r_{G\cup P}(e) = r_G(e)$ for all edges e on P, in $O(k(m + n \log n))$ time. ◀

4.6 2-hop cover

Let $G = (V, E)$ be a directed graph with an edge-weight function $w : E \to \mathbb{R}_{\ge 0}$. A *2-hop cover* of G is the following data structure (L^+, L^-) for efficiently answering distance queries. For each vertex $u \in V$, we assign a set $L^+(u)$ of pairs $(v, d^+_{uv}) \in V \times \mathbb{R}_{\ge 0}$ and a set $L^-(u)$ of pairs $(v, d^-_{vu}) \in V \times \mathbb{R}_{\ge 0}$. We require that, for every pair of vertices $s, t \in V$, the shortest-path distance from s to t is exactly the minimum of $d^+_{sh} + d^-_{ht}$ among all pairs $(h, d^+_{sh}) \in L^+(s)$

and $(h, d^-_{ht}) \in L^-(t)$. The *size* of the 2-hop cover is defined as $\sum_{u \in V} |L^+(u)| + |L^-(u)|$, and the *maximum label size* is defined as $\max_{u \in V} |L^+(u)| + |L^-(u)|$. Using a 2-hop cover of maximum label size T, we can answer a distance query in $O(T)$ time. In this section, we prove the following theorem.

▶ **Theorem 14.** *Given a non-negative edge-weighted directed graph and its elimination forest of depth k, we can construct a 2-hop cover of maximum label size $2k$ in $O(k(m + n \log n))$ time.*

For $S \subseteq V$, we define $f(S)$ as a function that returns a 2-hop cover of $G[S]$. We denote the shortest-path distance from s to t in $G[S]$ by $d_S(s, t)$. We denote the result of the distance query from s to t for $f(S)$ by $q_S(s, t)$. We now describe Increment and Union.

Increment$(X, f(X), x)$. Let (L^+, L^-) be the 2-hop cover of $G[X]$. By running Dijkstra's algorithm twice, we compute the shortest-path distances from x and to x in $G[X \cup \{x\}]$. Then, for each $u \in X \cup \{x\}$, we insert $(x, d_{X \cup \{x\}}(u, x))$ into $L^+(u)$ and $(x, d_{X \cup \{x\}}(x, u))$ into $L^-(u)$. Finally, we return the updated (L^+, L^-) as $f(X \cup \{x\})$.

▶ **Claim 2.** *$f(X \cup \{x\})$ is a 2-hop cover of $G[X \cup \{x\}]$.*

Proof. It suffices to show that $q_{X \cup \{x\}}(s, t) = d_{X \cup \{x\}}(s, t)$ holds for every $s, t \in X \cup \{x\}$. The claim clearly holds when $s = x$ or $t = x$. For $s, t \in X$, let $\delta := d_{X \cup \{x\}}(s, x) + d_{X \cup \{x\}}(x, t)$. Then, we have $d_{X \cup \{x\}}(s, t) = \min(d_X(s, t), \delta)$. From the construction of $f(X \cup \{x\})$, we have $q_{X \cup \{x\}} = \min(q_X(s, t), \delta) = \min(d_X(s, t), \delta)$. Therefore, the claim holds. ◀

Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c)))$. Because there exist no paths connecting X_i and X_j for any $i \neq j$, we can construct a 2-hop cover of $G[\bigcup_i X_i]$ by simply concatenating the 2-hop covers $f(X_1), \ldots, f(X_c)$.

Proof of Theorem 14. The algorithm Increment$(X, f(X), x)$ correctly computes $f(X \cup \{x\})$ in $O(|E[X]| + |X| \log |X|)$ time and the algorithm Union$((X_1, f(X_1)), \ldots, (X_c, f(X_c))$ correctly computes $f(\bigcup_i X_i)$ in $O(|\bigcup_i X_i|)$ time. Therefore, from Theorem 2, we can compute a 2-hop cover in $O(k(m + n \log n))$ time. Let (L^+, L^-) be the 2-hop cover obtained by computing $f(V)$. For each element $(u, d^+_{uv}) \in L^+(u)$ or $(u, d^-_{vu}) \in L^-(u)$, v is located on the path from u to the root in the elimination forest. Therefore, we have $|L^+(u)| + |L^-(u)| \leq 2k$ for every vertex $u \in V$. ◀

5 Open problems

Is it possible to obtain a $\mathrm{tw}^{O(1)} m \operatorname{polylog}(n)$-time algorithm for the edge-disjoint maximum $s - t$ flow problem? Because it looks difficult to obtain a maximum flow for G from a maximum flow for $G - v$ in linear time, it will be difficult to apply our approach to this problem. Another open question is whether the running time for (unweighted) Maximum Matching is optimal. For this problem, as it can be solved in $O(\sqrt{n}m)$ time, our algorithm improves the general-case algorithm only when $\mathrm{td} = n^{\frac{1}{2} - \epsilon}$.

References

1 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *SODA*, pages 1681–1697, 2015.

2 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and Fixed Parameter Subquadratic Algorithms for Radius and Diameter in Sparse Graphs. In *SODA*, pages 377–391, 2016.

3 Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD*, pages 349–360, 2013.

4 Takuya Akiba, Christian Sommer, and Ken-ichi Kawarabayashi. Shortest-path queries for complex networks: exploiting low tree-width outside the core. In *EDBT*, pages 144–155, 2012.

5 Norbert Blum. A new approach to maximum matching in general graphs. In *ICALP*, pages 586–597, 1990.

6 Hans L. Bodlaender, Jitender S. Deogun, Klaus Jansen, Ton Kloks, Dieter Kratsch, Haiko Müller, and Zsolt Tuza. Rankings of Graphs. *SIAM J. Discrete Math.*, 11(1):168–181, 1998.

7 Shiva Chaudhuri and Christos D. Zaroliagis. Shortest paths in digraphs of small treewidth. part I: sequential algorithms. *Algorithmica*, 27(3):212–226, 2000.

8 Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and Distance Queries via 2-Hop Labels. *SIAM J. Comput.*, 32(5):1338–1355, 2003.

9 David Coudert, Guillaume Ducoffe, and Alexandru Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. *CoRR*, abs/1707.05016, 2017. URL: `http://arxiv.org/abs/1707.05016`.

10 Jack Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17(3):449–467, 1965.

11 Fedor V. Fomin, Daniel Lokshtanov, Michał Pilipczuk, Saket Saurabh, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. In *SODA*, pages 1419–1432, 2017.

12 Harold N. Gabow. Data Structures for Weighted Matching and Nearest Common Ancestors with Linking. In *SODA*, pages 434–443, 1990.

13 Harold N. Gabow. Data Structures for Weighted Matching and Extensions to b-matching and f-factors. *CoRR*, abs/1611.07541, 2016. URL: `http://arxiv.org/abs/1611.07541`.

14 Harold N. Gabow and Robert Endre Tarjan. A Linear-Time Algorithm for a Special Case of Disjoint Set Union. *J. Comput. Syst. Sci.*, 30(2):209–221, 1985.

15 Harold N. Gabow and Robert Endre Tarjan. Faster Scaling Algorithms for General Graph-Matching Problems. *J. ACM*, 38(4):815–853, 1991.

16 Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theor. Comput. Sci.*, 689:67–95, 2017.

17 Zvi Gotthilf and Moshe Lewenstein. Improved algorithms for the k simple shortest paths and the replacement paths problems. *Inf. Process. Lett.*, 109(7):352–355, 2009.

18 John Hershberger, Subhash Suri, and Amit M. Bhosle. On the difficulty of some shortest path problems. *ACM Trans. Algorithms*, 3(1):5:1–5:15, 2007.

19 David R. Karger, Daphne Koller, and Steven J. Phillips. Finding the hidden path: Time bounds for all-pairs shortest paths. *SIAM J. Comput.*, 22(6):1199–1217, 1993.

20 Meir Katchalski, William McCuaig, and Suzanne M. Seager. Ordered colourings. *Discrete Mathematics*, 142(1-3):141–154, 1995.

21 Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

22 Kavindra Malik, Ashok K. Mittal, and Santosh K. Gupta. The k most vital arcs in the shortest path problem. *Oper. Res. Lett.*, 8(4):223–227, 1989.

23 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. The power of data reduction for matching. *CoRR*, abs/1609.08879, 2016. URL: `http://arxiv.org/abs/1609.08879`.

24 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.

25 Jaroslav Nešetřil and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006.

26 James B. Orlin and Antonio Sedeño-Noda. An $O(nm)$ time algorithm for finding the min length directed cycle in a graph. In *SODA*, pages 1866–1879, 2017.

27 Léon Planken, Mathijs de Weerdt, and Roman van der Krogt. Computing all-pairs shortest paths by leveraging low treewidth. *J. Artif. Intell. Res.*, 43:353–388, 2012.

28 Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Trans. Algorithms*, 8(4):33:1–33:11, 2012.

29 Alejandro A. Schäffer. Optimal Node Ranking of Trees in Linear Time. *Inf. Process. Lett.*, 33(2):91–96, 1989.

30 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.

31 Vijay V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{V}E)$ general graph maximum matching algorithm. *Combinatorica*, 14(1):71–109, 1994.

32 Fang Wei. TEDI: efficient shortest path query answering on graphs. In *SIGMOD*, pages 99–110, 2010.

33 Virginia Vassilevska Williams and Ryan Williams. Subcubic Equivalences between Path, Matrix and Triangle Problems. In *FOCS*, pages 645–654, 2010.

A Unified Polynomial-Time Algorithm for Feedback Vertex Set on Graphs of Bounded Mim-Width

Lars Jaffke
Department of Informatics, University of Bergen, Norway
lars.jaffke@uib.no

O-joung Kwon
Logic and Semantics, Technische Universität Berlin, Berlin, Germany
ojoungkwon@gmail.com

Jan Arne Telle
Department of Informatics, University of Bergen, Norway
jan.arne.telle@uib.no

Abstract

We give a first polynomial-time algorithm for (Weighted) Feedback Vertex Set on graphs of bounded *maximum induced matching width* (mim-width). Explicitly, given a branch decomposition of mim-width w, we give an $n^{\mathcal{O}(w)}$-time algorithm that solves Feedback Vertex Set. This provides a unified algorithm for many well-known classes, such as Interval graphs and Permutation graphs, and furthermore, it gives the first polynomial-time algorithms for other classes of bounded mim-width, such as Circular Permutation and Circular k-Trapezoid graphs for fixed k. In all these classes the decomposition is computable in polynomial time, as shown by Belmonte and Vatshelle [Theor. Comput. Sci. 2013].

We show that powers of graphs of tree-width $w-1$ or path-width w and powers of graphs of clique-width w have mim-width at most w. These results extensively provide new classes of bounded mim-width. We prove a slight strengthening of the first statement which implies that, surprisingly, Leaf Power graphs which are of importance in the field of phylogenetic studies have mim-width at most 1. Given a tree decomposition of width $w-1$, a path decomposition of width w, or a clique-width w-expression of a graph G, one can for any value of k find a mim-width decomposition of its k-power in polynomial time, and apply our algorithm to solve Feedback Vertex Set on the k-power in time $n^{\mathcal{O}(w)}$.

In contrast to Feedback Vertex Set, we show that Hamiltonian Cycle is NP-complete even on graphs of linear mim-width 1, which further hints at the expressive power of the mim-width parameter.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Graph Width Parameters, Graph Classes, Feedback Vertex Set, Leaf Powers

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.42

Related Version A full version of the paper is available at `https://arxiv.org/abs/1710.07148v2`, [23]

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 42; pp. 42:1–42:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A feedback vertex set in a graph is a subset of its vertices whose removal results in an acyclic graph. The problem of finding a smallest such set is one of Karp's 21 famous NP-complete problems [26] and many algorithmic techniques have been developed to attack this problem, see e.g. the survey [14]. The study of Feedback Vertex Set through the lens of parameterized algorithmics dates back to the earliest days of the field [11] and throughout the years numerous efforts have been made to obtain faster algorithms for this problem [4, 8, 9, 10, 11, 12, 18, 25, 32, 33]. In terms of parameterizations by structural properties of the graph, Feedback Vertex Set is e.g. known to be FPT parameterized by tree-width [9] and clique-width [6], and W[1]-hard but in XP parameterized by Independent Set and the size of a maximum induced matching [24].

In this paper, we study Feedback Vertex Set parameterized by the *maximum induced matching width* (mim-width for short), a graph parameter defined in 2012 by Vatshelle [36] which measures how easy it is to decompose a graph along vertex cuts with bounded maximum induced matching size on the bipartite graph induced by edges crossing the cut. One interesting aspect of this width-measure is that its modeling power is much stronger than tree-width and clique-width, and many well-known and deeply studied graph classes such as Interval graphs and Permutation graphs have (linear) mim-width 1, with decompositions that can be found in polynomial time [1, 36], while their clique-width can be proportional to the square root of the number of vertices [17]. Hence, designing an algorithm for a problem Π that runs in XP time parameterized by mim-width yields polynomial-time algorithms for Π on several interesting graph classes at once.

We give an XP-time algorithm for Feedback Vertex Set parameterized by mim-width, assuming that a branch decomposition of bounded mim-width is given.[1] Since such a decomposition can be computed in polynomial time [1, 36] for the following classes, this provides a unified polynomial-time algorithm for Feedback Vertex Set on all of them: Interval and Bi-Interval graphs, Circular Arc, Permutation and Circular Permutation graphs, Convex graphs, k-Trapezoid, Circular k-Trapezoid, k-Polygon, Dilworth-k and Co-k-Degenerate graphs for fixed k. Furthermore, our algorithm can be applied to Weighted Feedback Vertex Set as well, which for several of these classes was not known to be solvable in polynomial time.

▶ **Theorem 1.** *Given an n-vertex graph and a branch decomposition of mim-width w, we can solve* (Weighted) Feedback Vertex Set *in time $n^{\mathcal{O}(w)}$.*

We note that some of the above mentioned graph classes of bounded mim-width also have bounded asteroidal number, and a polynomial-time algorithm for Feedback Vertex Set on graphs of bounded asteroidal number was previously known due to Kratsch et al. [27]. However, our algorithm improves this result. For instance, k-Polygon graphs have mim-width at most $2k$ [1] and asteroidal number k [35]. The algorithm of Kratsch et al. [27] implies that Feedback Vertex Set on k-Polygon graphs can be solved in time $n^{\mathcal{O}(k^2)}$ while the our result improves this bound to $n^{\mathcal{O}(k)}$ time. It is not difficult to see that in general, mim-width and asteroidal number are incomparable.

We give results that expand our knowledge of the expressive power of mim-width. The k-power of a graph is the graph obtained by adding an edge vw for two vertices v, w with distance at most k. We show that powers of graphs of tree-width $w-1$ or path-width w and powers of graphs of clique-width w have mim-width at most w.

[1] This problem was mentioned as an 'interesting topic for further research' in [24]. Furthermore, the authors recently proved it to be W[1]-hard [21].

▶ **Theorem 2.** *Given a nice tree decomposition of width w, all of whose join bags have size at most w, or a clique-width w-expression of a graph, one can output a branch decomposition of mim-width w of its k-power in polynomial time.*

Theorem 2 implies that LEAF POWER graphs, of importance in the field of phylogenetic studies, have mim-width 1. These graphs are known to be STRONGLY CHORDAL and there has recently been interest in delineating the difference between these two graph classes, on the assumption that this difference was not very big [28, 30]. Our result actually implies a large difference, as it was recently shown by Mengel that there are STRONGLY CHORDAL SPLIT graphs of mim-width linear in the number of vertices [29].

We contrast our positive result with a proof that HAMILTONIAN CYCLE is NP-complete on graphs of linear mim-width 1, even when given a decomposition. Panda and Pradhan [31] showed that HAMILTONIAN CYCLE is NP-complete on ROOTED DIRECTED PATH graphs and we show that the graphs constructed in their reduction have linear mim-width 1. This provides evidence that the class of graphs of linear mim-width 1 is larger than one might have previously expected. Up until now, on all graph classes of linear mim-width 1, HAMILTONIAN CYCLE was known to be polynomial time (PERMUTATION), or even linear time (INTERVAL) solvable. This can be compared with the fact that parameterized by clique-width, FEEDBACK VERTEX SET is FPT [6] and HAMILTONIAN CYCLE only admits an XP algorithm [3, 13] but is W[1]-hard [15] (see also [16]).

Let us explain some of the essential ingredients of our dynamic programming algorithm. A crucial observation is that if a forest contains no induced matching of size $w+1$, then the number of internal vertices of the forest is bounded by $6w$ (Lemma 7). Motivated by this observation, given a forest, we define the forest obtained by removing its isolated vertices and leaves to be its *reduced forest*. The observation implies that in a cut (A, B) of a graph G, there are at most n^{6w} possible reduced forests of some forests consisting of edges crossing this cut. We enumerate all of them, and use these as indices of the table of our algorithm.

However, the interaction of an induced forest F in G with the edges of the bipartite graph crossing the cut (A, B), denote this graph by $G_{A,B}$, is not completely described by its reduced forest R. Observe that there might still be edges in the graph $G_{A,B}$ after removing the vertices of R; however, these edges are not contained in the forest F. We capture this property of F by considering a minimal vertex cover of $G_{A,B} - V(R)$ that avoids all vertices in F. Hence, as a second component of the table indices, we enumerate all minimal vertex covers of $G_{A,B} - V(R)$, for any possible reduced forest R.

To argue that the number of table entries stays bounded by $n^{\mathcal{O}(w)}$, we use the known result that every n-vertex bipartite graph with maximum induced matching size w has n^w minimal vertex covers. Remark that in the companion paper [22], we use minimal vertex covers of a bipartite graph in a similar way. The usage here is more complicated because we cannot index the table by the full intersection forest, but only index by its reduced forest.

Throughout the paper, proofs of statements marked with '★' are deferred to the full version [23].

2 Preliminaries

For a graph G we denote by $V(G)$ and $E(G)$ its vertex and edge set, respectively. For a vertex set $X \subseteq V(G)$, we denote by $G[X]$ the subgraph *induced* by X. We use the shorthand $G - X$ for $G[V(G) \setminus X]$. The union and intersection of two graphs G_1 and G_2 are denoted by $G_1 \cup G_2$ and $G_1 \cap G_2$, respectively. For a vertex $v \in V(G)$, we denote by $N_G(v)$ the set of *neighbors* of v in G. For $A \subseteq V(G)$, let $N_G(A)$ be the set of vertices in $V(G) \setminus A$ having a

neighbor in A. When G is clear from the context, we allow to remove it from the subscript. We denote by $\mathcal{C}(G)$ the set of connected components of G.

For disjoint vertex sets $X, Y \subseteq V(G)$, we denote by $G[X,Y]$ the bipartite subgraph of G with bipartition (X,Y) such that for $x \in X, y \in Y$, x and y are adjacent in $G[X,Y]$ if and only if they are adjacent in G. A *cut* of G is a bipartition (A,B) of its vertex set. A set M of edges is a *matching* if no two edges in M share an endpoint, and a matching $\{a_1b_1, \ldots, a_kb_k\}$ is *induced* if there are no other edges in the subgraph induced by $\{a_1, b_1, \ldots, a_k, b_k\}$. A vertex set $S \subseteq V(G)$ is a *vertex cover* of G if every edge in G is incident with a vertex in S.

For a graph G and a vertex subset A of G, we define $\mathrm{mim}_G(A)$ to be the maximum size of an induced matching in $G[A, V(G) \setminus A]$. A pair $(T, \mathcal{L})$ of a subcubic tree T and a bijection $\mathcal{L}$ from $V(G)$ to the set of leaves of T is called a *branch decomposition*. For each edge e of T, let T_1^e and T_2^e be the two connected components of $T-e$, and let (A_1^e, A_2^e) be the vertex bipartition of G such that for each $i \in \{1,2\}$, A_i^e is the set of all vertices in G mapped to leaves contained in T_i^e by $\mathcal{L}$. The *mim-width of* $(T, \mathcal{L})$ is defined as $\mathrm{mimw}(T, \mathcal{L}) := \max_{e \in E(T)} \mathrm{mim}_G(A_1^e)$. The minimum mim-width over all branch decompositions of G is called the *mim-width of* G and the *linear mim-width of* G if T is restricted to a path with a pendant leaf at each node.

Given a branch decomposition, one can subdivide an arbitrary edge and let the newly created vertex be the root of T, in the following denoted by r. Throughout the following we assume that each branch decomposition has a root node of degree two. For two nodes $t, t' \in V(T)$, we say that t' is a *descendant* of t if t lies on the path from r to t' in T. For $t \in V(T)$, we denote by G_t the subgraph induced by all vertices that are mapped to a leaf that is a descendant of t. We use the shorthand V_t for $V(G_t)$ and let $\bar{V_t} := V(G) \setminus V_t$.

▶ **Definition 3** (Boundary)**.** Let G be a graph and $A, B \subseteq V(G)$ such that $A \cap B = \emptyset$. We let $\mathrm{bd}_B(A)$ be the set of vertices in A that have a neighbor in B, i.e. $\mathrm{bd}_B(A) := \{v \in V(A) \mid N(v) \cap B \neq \emptyset\}$. We define $\mathrm{bd}(A) := \mathrm{bd}_{V(G)\setminus A}(A)$ and call $\mathrm{bd}(A)$ the *boundary* of A in G.

▶ **Definition 4** (Crossing Graph)**.** Let G be a graph and $A, B \subseteq V(G)$. If $A \cap B = \emptyset$, we define the graph $G_{A,B} := G[\mathrm{bd}_B(A), \mathrm{bd}_A(B)]$ to be the *crossing graph from* A *to* B.

For a node t in a branch decomposition, we define $G_{t,\bar{t}} := G_{V_t, \bar{V_t}}$.

We prove that given a set $A \subseteq V(G)$, the number of minimal vertex covers in $G_{A,V(G)\setminus A}$ is bounded by $n^{\mathrm{mim}_G(A)}$, and furthermore, the set of all minimal vertex covers can be enumerated in time $n^{\mathcal{O}(\mathrm{mim}_G(A))}$. This observation is crucial to argue that we only need to store $n^{\mathcal{O}(w)}$ entries at each node in the branch decomposition in our algorithm.

▶ **Corollary 5** (Minimal Vertex Covers Lemma, ★)**.** *Let H be a bipartite graph on n vertices with a bipartition (A,B). The number of minimal vertex covers of H is at most $n^{\mathrm{mim}_H(A)}$, and the set of all minimal vertex covers of H can be enumerated in time $n^{\mathcal{O}(\mathrm{mim}_H(A))}$.*

3 Reduced forests

We formally introduce the notion of a *reduced forest* which will be crucial to obtain the desired runtime bound of the algorithm for Feedback Vertex Set.

▶ **Definition 6** (Reduced Forest)**.** Let F be a forest. A *reduced forest* of F, denoted by $\mathfrak{R}(F)$, is an induced subforest of F obtained as follows. (i) Remove all isolated vertices of F. (ii) For each component C of F with $|V(C)| = 2$, remove one of its vertices. (iii) For each component C of F with $|V(C)| \geq 3$, remove all leaves of C.

Note that if F has no component that is a single edge then the reduced forest is uniquely defined. We give an upper bound on the size of a reduced forest $\mathfrak{R}(F)$ by a function of the size of a maximum induced matching in the forest F. This is crucial in our algorithm.

▶ **Lemma 7.** *Let p be a positive integer. If F is a forest whose maximum induced matching has size at most p and F' is a reduced forest of F, then $|V(F')| \leq 6p$.*

Proof. We sketch the proof, and provide the details in the full version [23]. For a forest F, we denote by $m(F)$ the size of the maximum induced matching in F. We prove by induction on $m(F)$. We may assume F contains no isolated vertices. If $m(F) \leq 1$, then F consists of one component containing no path of length 4, and $\mathfrak{R}(F)$ contains at most 2 nodes. We may assume $m(F) = p > 1$.

If F contains a connected component C containing no path of length 4, then it contains at most 2 internal nodes, and $m(F - V(C)) = m(F) - 1$. We may assume every component C of F contains a path of length 4. Assume F contains a path $v_1v_2v_3v_4v_5$ such that v_1 and v_5 are not leaves of F, and v_2, v_3, v_4 have degree 2 in $\mathfrak{R}(F)$. In this case, we define F' as the forest obtained from F by removing v_2, v_3, v_4 and adding an edge v_1v_5. Then we have $m(F') \leq m(F) - 1$. By induction hypothesis, $\mathfrak{R}(F')$ contains at most $6(p-1)$ nodes, and thus $\mathfrak{R}(F)$ contains at most $6(p-1)+3 \leq 6p$ nodes. We may assume there is no such a path.

Let C be a component of F. As $\mathfrak{R}(C)$ contains at least 3 nodes, the leaves of $\mathfrak{R}(C)$ form an independent set. Suppose $\mathfrak{R}(C)$ contains t leaves. Since each leaf of $\mathfrak{R}(C)$ is incident with a leaf of C, $\mathfrak{R}(C)$ contains an induced matching of size at least t. Thus, $m(F - V(C)) \leq m(F) - t$. Note that $\mathfrak{R}(C)$ contains at most t nodes of degree at least 3. By the previous argument, there are at most 2 nodes between two nodes of degree other than 2 in $\mathfrak{R}(C)$. Thus, $\mathfrak{R}(C)$ contains at most $t + t + 2(2t-1) \leq 6t$ nodes. The result follows by induction hypothesis. ◀

Let (A, B) be a vertex partition of a graph G, and R be some forest in $G_{A,B}$. In the algorithm, we will be asking if there exists an induced forest F in $G[A \cup \mathrm{bd}(B)]$ such that $F \cap G_{A,B}$ has R as a reduced forest. However, this formulation turns out to be technical, as we need to significantly consider some edges in B when we merge two partial solutions. To ease this task, we define the following notion on an induced forest in $G[A \cup \mathrm{bd}(B)] - E(G[\mathrm{bd}(B)])$.

▶ **Definition 8** (Forest respecting a forest and a minimal vertex cover)**.** Let (A, B) be a vertex partition of a graph G. Let R be an induced forest in $G_{A,B}$ and M be a minimal vertex cover of $G_{A,B} - V(R)$. An induced forest F in $G[A \cup \mathrm{bd}(B)] - E(G[\mathrm{bd}(B)])$ *respects* (R, M) if it satisfies the following: (i) R is a reduced forest of $F \cap G_{A,B}$ and (ii) $V(F) \cap M = \emptyset$.

Suppose R is an induced forest in $G_{A,B}$. For an induced forest F of G containing $V(R)$, there are two necessary conditions for R to be a reduced forest of $F \cap G_{A,B}$. First, if $F \cap G_{A,B}$ contains a vertex v in $G_{A,B} - V(R)$ having at least two neighbors in R, then v should be contained in the reduced forest. Therefore, in $F \cap G_{A,B}$, every vertex in $V(F \cap G_{A,B}) \setminus V(R)$ should have at most one neighbor in R. Second, every leaf x of R should have a neighbor y in $G_{A,B} - V(R)$ such that the only neighbor of y in R is x; otherwise, we would have removed x when taking a reduced forest. Motivated by this observation we define the notion of potential leaves, which is a possible leaf neighbor of some vertex in $V(R)$.

▶ **Definition 9** (Potential Leaves)**.** Let (A, B) be a vertex partition of a graph G. Let R be an induced forest in $G_{A,B}$ and M be a minimal vertex cover of $G_{A,B} - V(R)$. Let $H := G_{A,B}$. For each vertex $x \in V(R)$, we define its set of *potential leaves* as $PL_{R,M}(x) := N_H(x) \setminus N_H(V(R) \setminus \{x\}) \setminus (M \cup V(R))$.

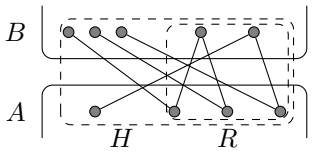


Figure 1 The graph R is a reduced forest of H.

For a subset A' of A, we consider a pair of an induced forest R' and a minimal vertex cover M' of $G_{A',V(G)\setminus A'} - V(R')$ and we say that this pair is a restriction of a pair of R and M for A, if they satisfy certain natural properties. In the dynamic programming, this will be necessary when considering cuts corresponding to some node and its child.

▶ **Definition 10** (Restriction of a reduced forest and a minimal vertex cover)**.** Let (A_1, A_2, B) be a vertex partition of a graph G. Let R be an induced forest in $G_{A_1\cup A_2,B}$ and M be a minimal vertex cover of $G_{A_1\cup A_2,B} - V(R)$. An induced forest R_1 in $G_{A_1,A_2\cup B}$ and a minimal vertex cover M_1 of $G_{A_1,A_2\cup B} - V(R_1)$ are *restrictions* of R and M to $G_{A_1,A_2\cup B}$ if they satisfy the following:

1. $V(R) \cap A_1 \subseteq V(R_1)$ and for every $v \in V(R) \cap B$ having at least two neighbors in $V(R) \cap A_1$, $v \in V(R_1)$.
2. $(V(R_1) \setminus V(R)) \cap B = \emptyset$ and $V(R_1) \cap M = \emptyset$.
3. Every vertex in $(V(R_1) \setminus V(R)) \cap A_1$ has at most one neighbor in $V(R) \cap B$.
4. $V(R) \cap M_1 = \emptyset$ and $M \cap A_1 \subseteq M_1$.
5. Let v be a vertex in $M \cap B$ incident with an edge vw in $G_{A_1,B} - V(R)$ for some $w \notin V(R_1)$ that is not covered by any vertices in $M \setminus \{v\}$. Then either $v \in M_1$ or $w \in M_1$.

Lastly, we define a notion for merging two partial solutions.

▶ **Definition 11** (Compatibility)**.** Let (A_1, A_2, B) be a vertex partition of a graph G. Let R be an induced forest in $G_{A_1\cup A_2,B}$, and for each $i \in \{1, 2\}$, let R_i be an induced forest in $G_{A_i,A_{3-i}\cup B}$, and P_i be a partition of $\mathcal{C}(R_i)$. We construct an auxiliary graph Q with respect to (R, R_1, R_2, P_1, P_2) in G as follows. Let Q be the graph on $\mathcal{C}(R) \cup \mathcal{C}(R_1) \cup \mathcal{C}(R_2)$ such that

- for H_1 and H_2 contained in distinct sets of $\mathcal{C}(R), \mathcal{C}(R_1), \mathcal{C}(R_2)$, H_1 is adjacent to H_2 in Q iff $V(H_1) \cap V(H_2) \neq \emptyset$,
- for $H_1, H_2 \in \mathcal{C}(R_i)$, H_1 is adjacent to H_2 iff they are contained in the same part of P_i,
- $\mathcal{C}(R)$ is an independent set.

We say that the tuple (R, R_1, R_2, P_1, P_2) is *compatible* in G if Q has no cycles. We define $\mathcal{U}(R, R_1, R_2, P_1, P_2)$ to be the partition of $\mathcal{C}(R)$ such that for $H_1, H_2 \in \mathcal{C}(R)$, H_1 and H_2 are contained in the same part iff they are contained in the same component of Q.

▶ **Proposition 12.** *Let (A_1, A_2, B) be a vertex partition of a graph G. Let R be an induced forest in $G_{A_1\cup A_2,B}$ and M be a minimal vertex cover of $G_{A_1\cup A_2,B} - V(R)$. Let H be an induced forest in $G[A_1 \cup A_2 \cup \mathrm{bd}(B)] - E(G[\mathrm{bd}(B)])$ respecting (R, M). There are restrictions (R_1, M_1) and (R_2, M_2) of (R, M) to $G_{A_1,A_2\cup B}$ and $G_{A_2,A_1\cup B}$, respectively such that*

- *for each $i \in \{1, 2\}$, $H \cap G[A_i \cup \mathrm{bd}(A_{3-i} \cup B)] - E(G[\mathrm{bd}(A_{3-i} \cup B)])$ respects (R_i, M_i),*
- *every vertex in $(V(R) \setminus (V(R_1) \cup V(R_2))) \cap B$ has at least two neighbors in $(V(R_1) \cap A_1) \cup (V(R_2) \cap A_2)$.*

Proof. For each $i \in \{1, 2\}$, let $F_i^* := H \cap G[A_i \cup \mathrm{bd}(A_{3-i} \cup B)] - E(G[\mathrm{bd}(A_{3-i} \cup B)])$, and $F_i := F_i^* \cap G_{A_i,A_{3-i}\cup B}$, and R_i be a reduced forest of F_i such that (Single-edge Rule) for a single-edge component vw of F_i with $v \in V(R)$ and $w \notin V(R)$, we select v as a vertex of R_i.

We check that every vertex in $S := (V(R) \setminus (V(R_1) \cup V(R_2))) \cap B$ has at least two neighbors in $(V(R_1) \cap A_1) \cup (V(R_2) \cap A_2)$. Suppose there exists a vertex v in S violating the condition. As $v \in V(R)$, v has at least two neighbors in $V(H) \cap (A_1 \cup A_2)$. Thus, v has a neighbor not contained in $(V(R_1) \cap A_1) \cup (V(R_2) \cap A_2)$. Let w be such a vertex, and without loss of generality, we assume $w \in A_1$. If v has a neighbor other than w in $V(H) \cap A_1$, then v is contained in R_1. So, in H, w is the unique neighbor of v in $V(H) \cap A_1$. Also, since $w \notin V(R_1)$, v is the unique neighbor of v in F_1. Then vw is a single-edge component of F_1, and by Single-edge Rule, we selected v as a vertex of R_1. This contradicts $v \notin V(R_1)$. We conclude that every vertex in S has at least two neighbors in $(V(R_1) \cap A_1) \cup (V(R_2) \cap A_2)$.

Conditions 1, 2 and 3 of being a restriction follows from the definition of a restriction and the Single-edge Rule. The details are given in the full version [23].

We now construct a minimal vertex cover M_1 of $G_{A_1,A_2 \cup B} - V(R_1)$, and verify the fourth and fifth conditions of being a restriction. Let M' be the set of all vertices v in M incident with an edge vw in $G_{A_1,A_2} - V(R)$ where vw is not covered by $M \setminus \{v\}$ and $w \notin V(R_1)$.

▶ **Claim 13.** *There is a minimal vertex cover M_1 of $G_{A_1,A_2 \cup B} - V(R_1)$ satisfying the following.*

- *$V(R) \cap M_1 = \emptyset$ and $M \cap A_1 \subseteq M_1$.*
- *Let v be a vertex in $M \cap B$ incident with an edge vw in $G_{A_1,B} - V(R)$ for some $w \notin V(R_1)$ that is not covered by any vertices in $M \setminus \{v\}$. Then either $v \in M_1$ or $w \in M_1$.*

Proof. Let Y be the set of all vertices in $\mathrm{bd}(A_2) \setminus V(H)$ having a neighbor in $\mathrm{bd}(A_1) \setminus V(R_1)$. Let Z be the set of all vertices in $\mathrm{bd}(A_1) \setminus V(R_1) \setminus (M \cap A_1)$ having a neighbor in $(V(R) \setminus V(R_1)) \cap B$. Let M'' be the set obtained from $M' \cup Y \cup Z$ by removing all vertices $v \in M' \cap B$ such that all the neighbors of v in $\mathrm{bd}(A_1) \setminus V(R_1) \setminus (M \cap A_1)$ are contained in Z.

By construction we can show that M'' is a vertex cover of $G_{A_1,A_2 \cup B} - V(R_1)$. We take a minimal vertex cover M_1 of $G_{A_1,A_2 \cup B} - V(R_1)$ contained in M''. We have $V(R) \cap M_1 = \emptyset$. Since each vertex of $M' \cap A$ covers some edge that is not covered by any other vertex in M'', we have $M \cap A_1 = M' \cap A_1 \subseteq M_1$. Since every vertex in Z meets some edge incident with $V(R) \setminus V(R_1)$, Z is contained in M_1. If v is a vertex in $M \cap B$ incident with an edge vw in $G_{A_1,B} - V(R)$ for some $w \notin V(R_1)$ that is not covered by any vertices in $M \setminus \{v\}$, then $v \in M' \cap B$. By construction of M'', either $v \in M'' \cap B$ or $w \in Z$. In particular if $w \notin Z$, then v is the vertex covering the edge vw, and it also remains in M_1. Thus, the fifth condition for being a restriction also holds, as required. ⌟

By Claim 13 we know that Conditions 4 and 5 of being a restriction hold, so we conclude that there is a restriction (R_1, M_1) of (R, M) where F_1^* respects (R_1, M_1). ◀

▶ **Proposition 14 (★).** *Let (A_1, A_2, B) be a vertex partition of a graph G. Let R be an induced forest in $G_{A_1 \cup A_2,B}$ and M be a minimal vertex cover of $G_{A_1 \cup A_2,B} - V(R)$. Let H be an induced forest in $G[A_1 \cup A_2 \cup \mathrm{bd}(B)] - E(G[\mathrm{bd}(B)])$ respecting (R, M) and for each $i \in \{1, 2\}$,*

- *let (R_i, M_i) be a restriction of (R, M) that $H_i := H \cap G[A_i \cup \mathrm{bd}(A_{3-i} \cup B)] - E(G[\mathrm{bd}(A_{3-i} \cup B)])$ respects (guaranteed by Proposition 12), and*
- *let P_i be the partition of $\mathcal{C}(R_i)$ such that for $C, C' \in \mathcal{C}(R_i)$, C and C' are in the same part iff they are contained in the same connected component of H_i.*

Then (R, R_1, R_2, P_1, P_2) is compatible.

Now, we prove a proposition regarding the merging operation of two partial solutions. Unfortunately, when we have partial solutions H_1 and H_2 for A_1 and A_2, respectively, $G[V(H_1) \cup V(H_2) \cup V(R)]$ may not be a partial solution. One reason is that since $M_1 \cap B$

may differ from $M \cap B$, H_1 might contain some vertex in $(M \setminus M_1) \cap B$. To avoid a situation where such a vertex is in R_1, we require that $V(R_1) \cap M = \emptyset$ (this is already included in the condition of being a restriction). Thus, such a vertex will be a potential leaf of some vertex in R_1, and we could simply remove it to find a forest avoiding M. The second reason is that for some vertex of $V(R) \cap V(A_1)$, it might have a potential leaf in A_2, but not in B, and thus in $G[V(H_1) \cup V(H_2) \cup V(R)]$ this vertex may not have a potential leaf as a neighbor even if it has degree at most 1 in R. In this case, we can simply add one of the potential leaves.

▶ **Proposition 15.** *Let (A_1, A_2, B) be a vertex partition of a graph G. Let R be an induced forest in $G_{A_1 \cup A_2, B}$ and M be a minimal vertex cover of $G_{A_1 \cup A_2, B} - V(R)$ such that for every vertex x of degree at most 1 in R, $PL_{R,M}(x) \neq \emptyset$. For each $i \in \{1, 2\}$,*

- *let R_i be an induced forest in $G_{A_i, A_{3-i} \cup B}$ and M_i be a minimal vertex cover of $G_{A_i, A_{3-i} \cup B} - V(R_i)$, and H_i be an induced forest in $G[A_i \cup \mathrm{bd}(A_{3-i} \cup B)] - E(G[\mathrm{bd}(A_{3-i} \cup B)])$ respecting (R_i, M_i),*
- *let P_i be the partition of $\mathcal{C}(R_i)$ such that for $C, C' \in \mathcal{C}(R_i)$, C and C' are in the same part if and only if they are contained in the same connected component of H_i,*
- *R_i and M_i are restrictions of R and M,*
- *every vertex in $(V(R)\ V(R_1) \cup V(R_2)) \cap B$ has at least two neighbors in $(V(R_1) \cap A_1) \cup (V(R_2) \cap A_2)$,*
- *(R, R_1, R_2, P_1, P_2) is compatible.*

There is an induced forest H in $G[A_1 \cup A_2 \cup \mathrm{bd}(B)] - E(G[\mathrm{bd}(B)])$ respecting (R, M) such that $V(H) \cap (A_1 \cup A_2) = (V(H_1) \cap A_1) \cup (V(H_2) \cap A_2)$.

Proof. As (R, R_1, R_2, P_1, P_2) is compatible, we can verify that $H^* := G[V(H_1) \cup V(H_2) \cup V(R)]$ is a forest. Let H be the graph obtained from $H^* - (B \setminus V(R))$ by adding a potential leaf of each vertex in $V(R) \cap (A_1 \cup A_2)$ of degree at most 1 in R and removing all edges between vertices in B. We observe that H is a forest. Since H^* is a forest, $H^* - (B \setminus V(R))$ is a forest. Adding a potential leaf of a vertex in $V(R) \cap (A_1 \cup A_2)$ preserves the property of being a forest, as we removed edges in $G[B]$. In the remainder, we prove that H respects (R, M); that is, (i) R is a reduced forest of $G_{A_1 \cup A_2, B} \cap H$, and (ii) $V(H) \cap M = \emptyset$.

Condition (ii) is easy to verify: since we remove all vertices in M when we construct H from H^*, we have $V(H) \cap M = \emptyset$. We now verify condition (i). Let $H_{new} := H \cap G_{A_1 \cup A_2, B}$. We first verify that every vertex of $V(H_{new}) \setminus V(R)$ has degree at most 1 in H_{new}.

▶ **Claim 16 (★).** *Every vertex of $V(H_{new}) \setminus V(R)$ has degree at most 1 in H_{new}.*

We argue that we can take R as a reduced forest of H_{new}. Let $v \in V(R)$. If v has degree at least 2 in H_{new}, then v is contained in any reduced forest of H_{new}. Suppose v has degree at most 1 in H_{new}. Suppose $v \in A_1 \cup A_2$. In this case, by the construction, v is incident with its potential leaf in H_{new}, say w. It means that vw is a single-edge component in H_{new}, and we can take v as a vertex in R.

Now, suppose $v \in B$. First assume that $v \in V(R_i)$ for some $i \in \{1, 2\}$. If v has a neighbor in R_i, then it also has at least one potential leaf in $H_i \cap G_{A_i, A_{3-i} \cup B}$, and thus v has degree 2 in H_{new}, a contradiction. Thus, v has no neighbor in R_i, and has exactly one potential leaf, say w. By Claim 16, v is the unique neighbor of w in R, and thus vw is a single-edge component of H_{new}. Thus, we can take v as a vertex in R. Suppose $v \in (V(R) \setminus (V(R_1) \cup V(R_2))) \cap B$. Then by the precondition, it has at least two neighbors in $(V(R_1) \cap A_1) \cup (V(R_2) \cap A_2) \subseteq (V(H_1) \cap A_1) \cup (V(H_2) \cap A_2)$. Therefore, it is contained in any reduced forest of H_{new}. It shows that R is a reduced forest of H_{new}.

Note that for each $i \in \{1, 2\}$, $V(H_i) \cap A_i$ avoids $M \cap A_i$. Furthermore, when we construct H_{new}, we removed all vertices in $M \cap B$. Therefore, we have $V(H_{new}) \cap M = \emptyset$. ◀

4 Feedback Vertex Set on graphs of bounded mim-width

We give an algorithm that solves the Feedback Vertex Set problem on graphs on n vertices together with a branch decomposition of mim-width w in time $n^{\mathcal{O}(w)}$. We observe that given a graph G, a subset of its vertices $S \subseteq V(G)$ is by definition a feedback vertex set if and only if $G - S$, the induced subgraph of G on vertices $V(G) \setminus S$, is an induced forest. It is therefore readily seen that computing the minimum size of a feedback vertex set is equivalent to computing the maximum size of an induced forest, so in the remainder of this section we solve the following problem which is more convenient for our exposition.

Maximum Induced Forest/Mim-Width
Input: A graph G on n vertices, a branch decomposition $(T, \mathcal{L})$ of G, an integer k.
Parameter: $w := \mathrm{mimw}(T, \mathcal{L})$.
Question: Does G contain an induced forest of size at least $n - k$?

We solve the Maximum Induced Forest problem by bottom-up dynamic programming over $(T, \mathcal{L})$, the given branch decomposition of G, starting at the leaves of T. Let $t \in V(T)$ be a node of T. To motivate the table indices of the dynamic programming table, we now observe how a solution to Maximum Induced Forest, an induced forest F, interacts with the graph $G_{t+\mathrm{bd}} := G[V_t \cup \mathrm{bd}(\bar{V}_t)] - E(G[\mathrm{bd}(\bar{V}_t)])$. The intersection of F with $G_{t+\mathrm{bd}}$ is an induced forest which throughout the following we denote by $F_{t+\mathrm{bd}} := F[V(G_{t+\mathrm{bd}})]$. Since we want to bound the number of table entries by $n^{\mathcal{O}(w)}$, we have to focus in particular on the interaction of F with the crossing graph $G_{t,\bar{t}}$, denoted by $F_{t,\bar{t}} := F[V(G_{t,\bar{t}})]$.

However, it is not possible to enumerate all induced forests in a crossing graph as potential table indices: Consider for example a star on n vertices and the cut consisting of the central vertex on one side and the remaining vertices on the other side. This cut has mim-value 1 but it contains 2^n induced forests, since each vertex subset of the star induces a forest on the cut. The remedy for this issue are *reduced* forests, introduced in Section 3.

At each node $t \in V(T)$, we only consider reduced forests as possible indices for the table entries. By Lemma 7, the number of reduced forests in each cut of mim-value w is bounded by n^{6w}. We analyze the structure of $F_{t,\bar{t}}$ to motivate the objects that can be used to represent $F_{t,\bar{t}}$ in such a way that the number of all possible table entries is at most $n^{\mathcal{O}(w)}$.

The induced forest $F_{t,\bar{t}}$ has three types of vertices in $G_{t,\bar{t}}$: (1) The vertices of the reduced forest $\mathfrak{R}(F_{t,\bar{t}})$ of $F_{t,\bar{t}}$. (2) The leaves of the induced forest $F_{t,\bar{t}}$, denoted by $L(F_{t,\bar{t}})$. (3) Vertices in $F_{t,\bar{t}}$ that do not have a neighbor in $F_{t,\bar{t}}$ on the opposite side of the boundary, in the following called *non-crossing* vertices and denoted by $\mathrm{NC}(F_{t,\bar{t}})$.

As outlined above, the only type of vertices in $F_{t,\bar{t}}$ that will be used as part of the table indices are the vertices of a reduced forest of $F_{t,\bar{t}}$. Hence, we neither know about the leaves of $F_{t,\bar{t}}$ nor its non-crossing vertices upon inspecting this part of the index. Suppose $v \in (L(F_{t,\bar{t}}) \cup \mathrm{NC}(F_{t,\bar{t}})) \cap V_t$. Then, $F_{t,\bar{t}}$ does not use any vertex in $x \in (N(v) \cap \bar{V}_t) \setminus V(\mathfrak{R}(F_{t,\bar{t}}))$: If v is a leaf in $F_{t,\bar{t}}$, then the presence of the edge vx would make it a non-leaf vertex and if v is a non-crossing vertex, the presence of vx would make v a vertex incident to an edge of the forest crossing the cut. An analogous point can be made for a vertex in $(L(F_{t,\bar{t}}) \cup \mathrm{NC}(F_{t,\bar{t}})) \cap \bar{V}_t$. We capture this property of $F_{t,\bar{t}}$ by considering a minimal vertex cover of $G_{t,\bar{t}} - V(\mathfrak{R}(F_{t,\bar{t}}))$ that avoids all leaves and non-crossing vertices of $F_{t,\bar{t}}$. Such a minimal vertex cover always exists as $L(F_{t,\bar{t}}) \cup \mathrm{NC}(F_{t,\bar{t}})$ is an independent set in $G_{t,\bar{t}}$.

Lastly, we have to keep track of how the connected components of $F_{t,\bar{t}}$ (respectively, $\mathfrak{R}(F_{t,\bar{t}})$) are joined together via the forest $F_{t+\mathrm{bd}}$. This forest induces a partition of $\mathcal{C}(\mathfrak{R}(F_{t,\bar{t}}))$ in the following way: Two components $C_1, C_2 \in \mathcal{C}(\mathfrak{R}(F_{t,\bar{t}}))$ are in the same part of the partition if and only if C_1 and C_2 are contained in the same connected component of $F_{t+\mathrm{bd}}$.

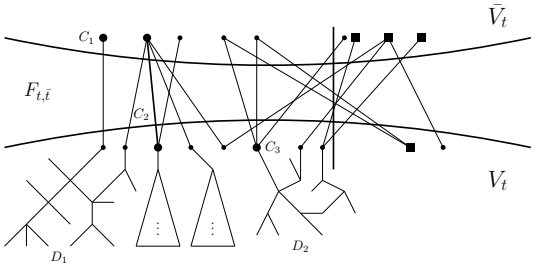


Figure 2 An example of a crossing graph $G_{t,\bar{t}}$ together with an induced forest F and their interaction. The forest $F_{t,\bar{t}} = F[V(G_{t,\bar{t}})]$ is displayed to the left of the dividing line in the drawing and the 4 vertices and 1 edge in bold form a reduced forest R of $F_{t,\bar{t}}$. The square vertices form a minimal vertex cover of $G_{t,\bar{t}} - V(R)$ satisfying (3). Furthermore, C_i ($i \in [3]$) are the connected components of R and D_i ($i \in [2]$) are the connected components of F.

We are ready to define the indices of the dynamic programming table $\mathcal{T}$ to keep track of sufficiently much information about the partial solutions in the graph $G_{t+\mathrm{bd}}$. We denote by $\mathcal{R}_t$ the set of all induced forests of $G_{t,\bar{t}}$ on at most $6w$ vertices. For $R \in \mathcal{R}_t$, let $\mathcal{M}_{t,R}$ be the set of all minimal vertex covers of $G_{t,\bar{t}} - V(R)$ and $\mathsf{P}_{t,R}$ the set of all partitions of the components of R. For an illustration of the definition of the table indices, which we start on now, see Figure 2. For $(R, M, P) \in \mathcal{R}_t \times \mathcal{M}_{t,R} \times \mathsf{P}_{t,R}$ and $i \in \{0, \ldots, n\}$, we set $\mathcal{T}[t, (R, M, P), i] := 1$ (and to 0 otherwise), iff the following conditions are satisfied.

1. There is an induced forest F in $G[V_t \cup \mathrm{bd}(\bar{V}_t)] - E(G[\mathrm{bd}(\bar{V}_t)])$ with $|V(F) \cap V_t| = i$.
2. Let $F_{t,\bar{t}} = F \cap G_{t,\bar{t}}$, i.e. $F_{t,\bar{t}}$ is the subforest of F induced by the vertices of the crossing graph $G_{t,\bar{t}}$. Then, $R = \mathfrak{R}(F_{t,\bar{t}})$, meaning that R is a reduced forest of $F_{t,\bar{t}}$.
3. M is a minimal vertex cover of $G_{t,\bar{t}} - V(R)$ such that $V(F) \cap M = \emptyset$.
4. P is a partition of $\mathcal{C}(R)$ such that two components $C_1, C_2 \in \mathcal{C}(R)$ are in the same part of the partition iff they are contained in the same connected component of F.

Recall that $r \in V(T)$ denotes the root of T, the tree of the given branch decomposition of G. From Property (1) we immediately observe that the table entries store enough information to obtain a solution to Maximum Induced Forest after all table entries have been filled. In other words, G contains an induced forest of size i if and only if $\mathcal{T}[r, (\emptyset, \emptyset, \emptyset), i] = 1$.

By definition, $|\mathcal{R}_t| = \mathcal{O}(n^{6w})$ and by Minimal Vertex Covers Lemma, $|\mathcal{M}_{t,R}| = n^{\mathcal{O}(w)}$ for each $R \in \mathcal{R}_t$. It is well known that $|\mathsf{P}_{t,R}| \leq (w/\log(w))^{\mathcal{O}(w)}$ by upper bounds on the Bell number (see e.g. [2]). Thus, we have

▶ **Proposition 17.** *There are at most $n^{\mathcal{O}(w)}$ table entries in $\mathcal{T}$.*

We now show how to compute the table entries in $\mathcal{T}$. We can easily fill in the table entries for the leaves of T, for the details see the full version [23]. Here, we focus on how to compute the entries in the internal nodes of T from the entries stored in the tables corresponding to their children. Let $t \in V(T)$ be an internal node with children a and b. Using Propositions 12, 14, 15, we can show the following.

▶ **Proposition 18 (★).** *Let $\mathfrak{I} = [(R, M, P), i] \in (\mathcal{R}_t \times \mathcal{M}_{t,R_t} \times \mathsf{P}_{t,R_t}) \times \{0, \ldots, n\}$ such that for every vertex x of degree at most 1 in R, $PL_{R,M}(x) \neq \emptyset$. Then $\mathcal{T}[t, (R, M, P), i] = 1$ if and only if there are restrictions (R_a, M_a) and (R_b, M_b) of (R, M) to $G_{a,\bar{a}}$ and $G_{b,\bar{b}}$, respectively, and partitions P_a and P_b of $\mathcal{C}(R_a)$ and $\mathcal{C}(R_b)$, respectively, and integers i_a and i_b such that*

- *$\mathcal{T}[t_a, (R_a, M_a, P_a), i_a] = 1$ and $\mathcal{T}[t_b, (R_b, M_b, P_b), i_b] = 1$,*
- *(R, R_a, R_b, P_a, P_b) is compatible and $P = \mathcal{U}(R, R_1, R_2, P_1, P_2)$,*

- *every vertex in* $(V(R) \setminus V(R_1) \cup V(R_2)) \cap B$ *has at least two neighbors in* $(V(R_1) \cap A_1) \cup (V(R_2) \cap A_2)$,
- $i_a + i_b = i$.

Based on Proposition 18, we can proceed with the computation of the table at an internal node t with children a and b. Let $\mathfrak{I} = [(R, M, P), i] \in (\mathcal{R}_t \times \mathcal{M}_{t,R_t} \times \mathsf{P}_{t,R_t}) \times \{0, \ldots, n\}$.

(Step 1) We verify whether $\mathfrak{I}$ is valid, i.e. whether it can represent a valid partial solution in the sense of the definition of the table entries. That is, each vertex of degree at most 1 in R has to have at least one potential leaf.

(Step 2) We consider all pairs $\mathfrak{I}_a = [(R_a, M_a, P_a), i_a] \in (\mathcal{R}_a \times \mathcal{M}_{a,R_a} \times \mathsf{P}_{a,R_a}) \times \{0, \ldots, n\}$ and $\mathfrak{I}_b = [(R_b, M_b, P_b), i_b] \in (\mathcal{R}_b \times \mathcal{M}_{b,R_b} \times \mathsf{P}_{b,R_b}) \times \{0, \ldots, n\}$. We check

- (R_a, M_a) and (R_b, M_b) are restrictions of (R, M) to $G_{a,\bar{a}}$ and $G_{b,\bar{b}}$ respectively,
- $\mathcal{T}[t_a, (R_a, M_a, P_a), i_a] = 1$ and $\mathcal{T}[t_b, (R_b, M_b, P_b), i_b] = 1$,
- (R, R_a, R_b, P_a, P_b) is compatible and $P = \mathcal{U}(R, R_1, R_2, P_1, P_2)$,
- every vertex in $(V(R) \setminus V(R_1) \cup V(R_2)) \cap B$ has at least two neighbors in $(V(R_1) \cap A_1) \cup (V(R_2) \cap A_2)$,
- $i_a + i_b = i$.

If there are $\mathfrak{I}_a$ and $\mathfrak{I}_b$ satisfying all of conditions, then we assign $\mathcal{T}[t, (R, M, P), i] = 1$, and otherwise, we assign $\mathcal{T}[t, (R, M, P), i] = 0$. Correctness follows from Proposition 18 and the runtime analysis is deferred to the full version [23].

In Weighted Feedback Vertex Set, we are given a graph and a function $\omega : V(G) \to \mathbb{R}$, we want to find a set S with minimum $\omega(S)$ such that $G - S$ has no cycles. Similar to Feedback Vertex Set, we can instead solve the problem of finding an induced forest F with maximum $\omega(V(F))$. Instead of specifying i in the table, for a table $[t, (R, M, P)]$ we keep the $\omega(V(F) \cap V_t)$ value for an induced forest F respecting (R, M) and P with maximum $\omega(V(F) \cap V_t)$, as $\mathcal{T}[t, (R, M, P)]$. The procedure for a leaf node is analogous. In the internal node, we compare all pairs (R_a, M_a, P_a) and (R_b, M_b, P_b) for children t_a and t_b, and take the maximum among all sums $\mathcal{T}[t_a, (R_a, M_a, P_a)] + \mathcal{T}[t_b, (R_b, M_b, P_b)]$. Therefore, we can solve Weighted Feedback Vertex Set in time $n^{\mathcal{O}(w)}$ as well. We have proved Theorem 1.

5 Hamiltonian Cycle for linear mim-width 1

▶ **Theorem 19.** Hamiltonian Cycle *is NP-complete on graphs of linear mim-width* 1*, even if given the mim-width decomposition.*

Proof. Itai et al [20] showed that given a bipartite graph G with maximum degree 3, it is NP-complete to decide if it has a Hamiltonian cycle, while Panda and Pradhan [31] construct, from this graph G, a rooted directed path graph H such that H has a Hamiltonian cycle if and only if G does. The construction of [31] can be used to also output a linear mim-width 1 decomposition of H, in polynomial time. We provide the details in the full version [23]. ◀

6 Powers of graphs

We show that k-powers of graphs of tree-width at most $w - 1$ have mim-width at most w. This is somewhat surprising because this bound does not depend on k. The following lemma captures the property. We denote by $\mathrm{dist}_G(v, w)$ the distance between v and w in G.

▶ **Lemma 20.** *Let* $k, w \in \mathbb{N}$ *and let* (A, B, C) *be a vertex partition of graph* G *such that there are no edges between* A *and* C*, and* B *has size* w*. If* H *is the* k*-power of* G*, then* $\mathrm{mim}_H(A \cup B) \leq w$.

Proof. Let $B := \{b_1, b_2, \ldots, b_w\}$. For every vertex v in G, we assign a vector $c_v = (c_1^v, \ldots, c_w^v)$ such that $c_i^v = \mathrm{dist}_G(v, b_i)$. Suppose for contradiction that there is an induced matching $\{y_1z_1, y_2z_2, \ldots, y_tz_t\}$ of size at least $w+1$ in $H[A \cup B, C]$. Since $t \geq w+1$, there are distinct integers $t_1, t_2 \in \{1, 2, \ldots, t\}$ and an integer $j \in \{1, 2, \ldots, w\}$ such that

- $\mathrm{dist}_G(y_{t_1}, b_j) + \mathrm{dist}_G(z_{t_1}, b_j) \leq k$ and $\mathrm{dist}_G(y_{t_2}, b_j) + \mathrm{dist}_G(z_{t_2}, b_j) \leq k$.

Then we have either $\mathrm{dist}_G(y_{t_1}, b_j) + \mathrm{dist}_G(z_{t_2}, b_j) \leq k$ or $\mathrm{dist}_G(y_{t_2}, b_j) + \mathrm{dist}_G(z_{t_1}, b_j) \leq k$, which contradicts with the assumption that $y_{t_1}z_{t_2}$ and $y_{t_2}z_{t_1}$ are not edges in H. ◀

▶ **Theorem 21 (★).** *Let $k, w \in \mathbb{N}$ and G be a graph that admits a nice tree decomposition of width w all of whose join bags are of size at most w. Then the k-power of G has mim-width at most w. Furthermore, given such a nice tree decomposition, we can output a branch decomposition of mim-width at most w in polynomial time.*

The following notions are of importance in the field of phylogenetic studies, i.e. the reconstruction of ancestral relations in biology, see e.g. [7]. A graph G is a leaf power if there exists a threshold k and a tree T, called a leaf root, whose leaf set is $V(G)$ such that $uv \in E$ if and only if the distance between u and v in T is at most k. Similarly, G is called a min-leaf power if $uv \in E$ if and only if the distance between u and v in T is more than k. Thus, G is a leaf power if an only if its complement is a min-leaf power. It is easy to see that trees admit nice tree decompositions all of whose join bags have size 1 and since every leaf power graph is an induced subgraph of a power of some tree, it has mim-width at most 1 by Theorem 21.

▶ **Corollary 22.** *The leaf powers and min-leaf powers have mim-width at most* 1 *and given a leaf root, we can compute in polynomial time a branch decomposition witnessing this.*

We further show that powers of graphs of clique-width w have mim-width at most w. We give the details of the proof in the full version [23]; however we remark that the following lemma will imply this result. A graph is *w-labeled* if there is a labeling function $f : V(G) \to \{1, 2, \ldots, w\}$.

▶ **Lemma 23 (★).** *Let $k, w \in \mathbb{N}$ and let (A, B) be a vertex partition of graph G such that $G[A]$ is w-labeled and two vertices in a label class of $G[A]$ have the same neighborhood in B. If H is the k-power of G, then* $\mathrm{mim}_H(A) \leq w$.

▶ **Theorem 24 (★).** *Let $k, w \in \mathbb{N}$ and G be a graph of clique-width w. Then the k-power of G has mim-width at most w. Furthermore, given a clique-width w-expression, we can output a branch decomposition of mim-width at most w in polynomial time.*

7 Conclusion

We have shown that Feedback Vertex Set admits an $n^{\mathcal{O}(w)}$-time algorithm when given with a branch decomposition of mim-width w. Our algorithm provides polynomial-time algorithms for known classes of bounded mim-width, and gives the first polynomial-time algorithms for Circular Permutation and Circular k-Trapezoid graphs for fixed k.

Somewhat surprisingly, we prove that powers of graphs of bounded tree-width or clique-width have bounded mim-width. Heggernes et al. [19] showed that the clique-width of the k-power of a path of length $k(k+1)$ is exactly k. This also shows that the expressive power of mim-width is much stronger than clique-width, since all powers of paths have mim-width just 1. As a special case, we show that Leaf Power graphs have mim-width 1. We believe the notion of mim-width can be of benefit to the study of Leaf Power graphs.

We conclude with repeating an open problem regarding algorithms for computing mim-width. The problem of computing the mim-width of general graphs was shown to be

W[1]-hard, not in APX unless NP = ZPP [34], and no algorithm for computing the mim-width of a graph in XP time is known. As in [34], we therefore ask: Is there an XP algorithm approximating mim-width w by some function $f(w)$ and returning a decomposition? We remark that it is a big open problem whether Leaf Power graphs can be recognized in polynomial time [5, 7, 28, 30]. A positive answer to our question may be used to design such a recognition algorithm using branch decompositions of bounded mim-width.

References

1. Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoretical Computer Science*, 511:54–65, 2013.
2. Daniel Berend and Tamir Tassa. Improved bounds on bell numbers and on moments of sums of random variables. *Probability and Mathematical Statistics*, 30(2):185–205, 2010.
3. Benjamin Bergougnoux, Mamadou Moustapha Kanté, and O-Joung Kwon. An optimal XP algorithm for hamiltonian cycle on graphs of bounded clique-width. In *Proceedings 15th International Symposium on Algorithms and Data Structures (WADS)*, pages 121–132, Cham, 2017. Springer.
4. Hans L. Bodlaender. On disjoint cycles. *International Journal of Foundations of Computer Science*, 5(1):59–68, 1994.
5. Andreas Brandstädt. On leaf powers. Technical report, University of Rostock, 2010.
6. Binh-Minh Bui-Xuan, Ondřej Suchỳ, Jan Arne Telle, and Martin Vatshelle. Feedback vertex set on graphs of low clique-width. *European Journal of Combinatorics*, 34(3):666–679, 2013.
7. Tiziana Calamoneri and Blerina Sinaimeri. Pairwise compatibility graphs: A survey. *SIAM Review*, 58(3):445–460, 2016. doi:10.1137/140978053.
8. Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *Journal of Computer and System Sciences*, 74(7):1188–1198, 2008.
9. Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Joham M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 150–159. IEEE, 2011.
10. Frank K. H. A. Dehne, Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Kim Stevens. An $\mathcal{O}(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. In *Proceedings 11th International Computing and Combinatorics Conference (COCOON)*, pages 859–869. Springer, 2005.
11. Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.
12. Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.
13. Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In *Proceedings 27th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 117–128. Springer, 2001.
14. Paola Festa, Panos M. Pardalos, and Mauricio G. C. Resende. Feedback set problems. In *Handbook of Combinatorial Optimization*, pages 209–258. Springer, 1999.
15. Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010.
16. Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM Journal on Computing*, 43(5):1541–1563, 2014.

17 Martin Charles Golumbic and Udi Rotics. On the clique-width of some perfect graph classes. *International Journal of Foundations of Computer Science*, 11(03):423–443, 2000.

18 Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *Journal of Computer and System Sciences*, 72(8):1386–1396, 2006.

19 Pinar Heggernes, Daniel Meister, Charis Papadopoulos, and Udi Rotics. Clique-width of path powers. *Discrete Applied Mathematics*, 205:62–72, 2016.

20 Alon Itai, Christos H. Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

21 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. A note on the complexity of feedback vertex set parameterized by mim-width. *arXiv preprints*, 2017. arXiv:1711.05157.

22 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Polynomial-time algorithms for the longest induced path and induced disjoint paths problems on graphs of bounded mim-width. In *Proceedings 12th International Symposium on Parameterized and Exact Computation (IPEC)*, pages 21:1–21:12, 2017. arXiv:1708.04536.

23 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. A unified polynomial-time algorithm for feedback vertex set on graphs of bounded mim-width. *arXiv preprints*, 2017. arXiv:1710.07148.

24 Bart M. P. Jansen, Venkatesh Raman, and Martin Vatshelle. Parameter ecology for feedback vertex set. *Tsinghua Science and Technology*, 19(4):387–409, 2014.

25 Iyad Kanj, Michael Pelsmajer, and Marcus Schaefer. Parameterized algorithms for feedback vertex set. In *Proceedings 1st International Workshop on Parameterized and Exact Computation (IWPEC)*, pages 235–247. Springer, 2004.

26 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

27 Dieter Kratsch, Haiko Müller, and Ioan Todinca. Feedback vertex set on at-free graphs. *Discrete Applied Mathematics*, 156(10):1936–1947, 2008.

28 Manuel Lafond. On strongly chordal graph that are not leaf powers. *arXiv preprints*, 2017. arXiv:1703.08018.

29 Stefan Mengel. Lower bounds on the mim-width of some perfect graph classes. *arXiv preprints*, 2016. arXiv:1608.01542, to appear in Discrete Applied Mathematics.

30 Ragnar Nevries and Christian Rosenke. Towards a characterization of leaf powers by clique arrangements. *Graphs and Combinatorics*, 32(5):2053–2077, 2016.

31 B. S. Panda and D. Pradhan. NP-completeness of hamiltonian cycle problem on rooted directed path graphs. *arXiv preprints*, 2008. arXiv:0809.2443.

32 Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for undirected feedback vertex set. In *Proceedings 13th International Symposium on Algorithms and Computation (ISAAC)*, pages 241–248. Springer, 2002.

33 Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for finding feedback vertex sets. *ACM Transactions on Algorithms*, 2(3):403–415, 2006.

34 Sigve H. Sæther and Martin Vatshelle. Hardness of computing width parameters based on branch decompositions over the vertex set. *Theoretical Computer Science*, 615:120–125, 2016.

35 Lorna Stewart and Richard Valenzano. On polygon numbers of circle graphs and distance hereditary graphs. *Discrete Applied Mathematics*, 2017. in press. `doi:10.1016/j.dam.2017.09.016`.

36 Martin Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, Norway, 2012.

Generalizing the Kawaguchi-Kyan Bound to Stochastic Parallel Machine Scheduling

Sven Jäger
Institut für Mathematik, Technische Universität Berlin, Germany
jaeger@math.tu-berlin.de

Martin Skutella
Institut für Mathematik, Technische Universität Berlin, Germany
skutella@math.tu-berlin.de

Abstract

Minimizing the sum of weighted completion times on m identical parallel machines is one of the most important and classical scheduling problems. For the stochastic variant where processing times of jobs are random variables, Möhring, Schulz, and Uetz (1999) presented the first and still best known approximation result, achieving, for arbitrarily many machines, performance ratio $1 + \frac{1}{2}(1 + \Delta)$, where Δ is an upper bound on the squared coefficient of variation of the processing times. We prove performance ratio $1 + \frac{1}{2}(\sqrt{2} - 1)(1 + \Delta)$ for the same underlying algorithm—the Weighted Shortest Expected Processing Time (WSEPT) rule. For the special case of deterministic scheduling (i.e., $\Delta = 0$), our bound matches the tight performance ratio $\frac{1}{2}(1+\sqrt{2})$ of this algorithm (WSPT rule), derived by Kawaguchi and Kyan in a 1986 landmark paper. We present several further improvements for WSEPT's performance ratio, one of them relying on a carefully refined analysis of WSPT yielding, for every fixed number of machines m, WSPT's exact performance ratio of order $\frac{1}{2}(1 + \sqrt{2}) - O(1/m^2)$.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization, Theory of computation → Scheduling algorithms

Keywords and phrases Stochastic Scheduling, Parallel Machines, Approximation Algorithm, List Scheduling, Weighted Shortest (Expected) Processing Time Rule

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.43

Related Version A full version of this paper can be found at `https://arxiv.org/abs/1801.01105`.

Funding This research was carried out in the framework of MATHEON and supported by the Einstein Foundation Berlin.

Acknowledgements We would like to thank the anonymous referees for careful reading and helpful comments.

1 Introduction

In an archetypal machine scheduling problem, n independent jobs have to be scheduled on m identical parallel machines or processors. Each job j is specified by its processing time $p_j > 0$ and by its weight $w_j > 0$. In a feasible schedule, every job j is processed for p_j time units on one of the m machines in an uninterrupted fashion, and every machine can process at most one job at a time. The completion time of job j in some schedule S is denoted by C_j^{S}. The goal is to compute a schedule S that minimizes the total weighted completion time

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 43; pp. 43:1–43:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$\sum_{j=1}^{n} w_j C_j^{\mathrm{S}}$. In the standard classification scheme of Graham, Lawler, Lenstra, and Rinnooy Kan [7], this NP-hard scheduling problem is denoted by $P||\sum w_j C_j$.

Weighted Shortest Processing Time Rule. By a well-known result of Smith [24], sequencing the jobs in order of non-increasing ratios w_j/p_j gives an optimal single-machine schedule. List scheduling in this order is known as the Weighted Shortest Processing Time (WSPT) rule and can also be applied to identical parallel machines, where it is a $\frac{1}{2}(1+\sqrt{2})$-approximation algorithm; see Kawaguchi and Kyan [14]. A particularly remarkable aspect of Kawaguchi and Kyan's work is that, in contrast to the vast majority of approximation results, their analysis does not rely on some kind of lower bound. Instead, they succeed in explicitly identifying a class of worst-case instances. In particular, the performance ratio $\frac{1}{2}(1+\sqrt{2})$ is tight: For every $\varepsilon > 0$ there is a problem instance for which WSPT has approximation ratio at least $\frac{1}{2}(1+\sqrt{2}) - \varepsilon$. The instances achieving these approximation ratios, however, have large numbers of machines when ε becomes small. Schwiegelshohn [20] gives a considerably simpler version of Kawaguchi and Kyan's analysis.

Stochastic Scheduling. Many real-world machine scheduling problems exhibit a certain degree of uncertainty about the jobs' processing times. This characteristic is captured by the theory of stochastic machine scheduling, where the processing time of job j is no longer a given number p_j but a random variable $\boldsymbol{p}_j$. As all previous work in the area, we always assume that these random variables are stochastically independent. At the beginning, only the distributions of these random variables are known. The actual processing time of a job becomes only known upon its completion. As a consequence, the solution to a stochastic scheduling problem is no longer a simple schedule, but a so-called *non-anticipative scheduling policy*. Precise definitions on stochastic scheduling policies are given by Möhring, Radermacher, and Weiss [16]. Intuitively, whenever a machine is idle at time t, a non-anticipative scheduling policy may decide to start a job of its choice based on the observed past up to time t as well as the a priori knowledge of the jobs processing time distributions and weights. It is, however, not allowed to anticipate information about the future, i.e., the actual realizations of the processing times of jobs that have not yet finished by time t.

It follows from simple examples that, in general, a non-anticipative scheduling policy cannot yield an optimal schedule for each possible realization of the processing times. We are therefore looking for a policy which minimizes the objective in expectation. For the stochastic scheduling problem considered in this paper, the goal is to find a non-anticipative scheduling policy that minimizes the expected total weighted completion time. This problem is denoted by $P|\boldsymbol{p}_j \sim \mathrm{stoch}|\mathrm{E}[\sum w_j \boldsymbol{C}_j]$.

Weighted Shortest Expected Processing Time Rule. The stochastic analogue of the WSPT rule is greedily scheduling the jobs in order of non-increasing ratios $w_j/\mathrm{E}[\boldsymbol{p}_j]$. Whenever a machine is idle, the Weighted Shortest Expected Processing Time (WSEPT) rule immediately starts the next job in this order. For a single machine this is again optimal; see Rothkopf [18]. For identical parallel machines, Cheung, Fischer, Matuschke, and Megow [3] and Im, Moseley, and Pruhs [10] independently show that WSEPT does not even achieve constant performance ratio. More precisely, for every $R > 0$ there is a problem instance for which WSEPT's expected total weighted completion time is at least R times the expected objective value of an optimal non-anticipative scheduling policy. In the special case of exponentially distributed processing times, Jagtenberg, Schwiegelshohn, and Uetz [12] show a lower bound of 1.243 on WSEPT's performance. On the positive side, WSEPT is an optimal

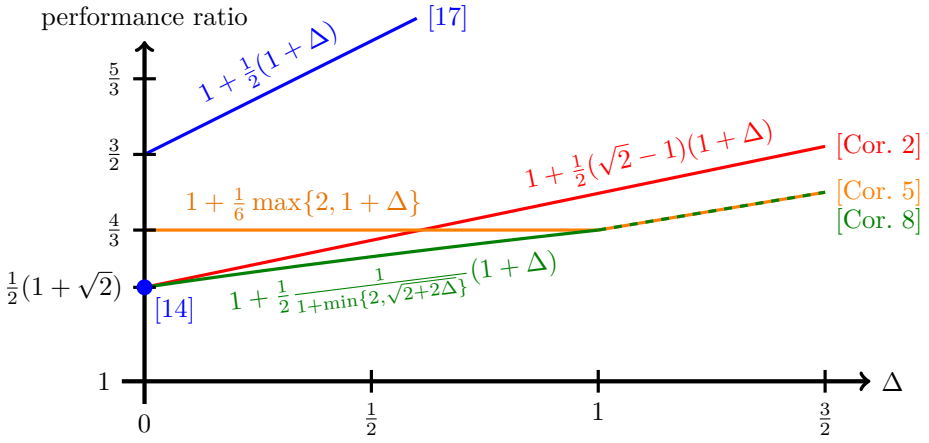


Figure 1 Bounds on WSEPT's performance ratio.

policy for the special case of unit weight jobs with stochastically ordered processing times, $P|\boldsymbol{p}_j \sim \text{stoch}(\preceq_{\text{st}})|E[\sum \boldsymbol{C}_j]$; see Weber, Varaiya, and Walrand [25]. Moreover, Weiss [26, 27] proves asymptotic optimality of WSEPT for bounded second moments of the residual processing time distributions. Möhring, Schulz, and Uetz [17] show that WSEPT achieves performance ratio $1+\frac{1}{2}(1+\Delta)(1-\frac{1}{m})$, where Δ is an upper bound on the squared coefficient of variation of the processing times.

Further Approximation Results from the Literature. While there is a PTAS for the deterministic problem $P||\sum w_j C_j$ [23], no constant-factor approximation algorithm is known for the stochastic problem $P|\boldsymbol{p}_j \sim \text{stoch}|\text{E}[\sum w_j \boldsymbol{C}_j]$. WSEPT's performance ratio $1+\frac{1}{2}(1+\Delta)$ (for arbitrarily many machines) proven by Möhring et al. [17] is the best hitherto known performance ratio. The only known approximation ratio not depending on the jobs' squared coefficient of variation Δ is due to Im et al. [10], who, for the special case of unit job weights $P|\boldsymbol{p}_j \sim \text{stoch}|\text{E}[\sum \boldsymbol{C}_j]$, present an $O(\log^2 n + m \log n)$-approximation algorithm.

The performance ratio $1+\frac{1}{2}(1+\Delta)$ has been carried over to different generalizations of $P|\boldsymbol{p}_j \sim \text{stoch}|\text{E}[\sum w_j \boldsymbol{C}_j]$. Megow, Uetz, and Vredeveld [15] show that it also applies if jobs arrive online in a list and must immediately and irrevocably be assigned to machines, on which they can be sequenced optimally. An approximation algorithm with this performance ratio for the problem on unrelated parallel machines is designed by Skutella, Sviridenko, and Uetz [22]. If these two features are combined, i.e., in the online list-model with unrelated machines, Gupta, Moseley, Uetz, and Xie [8] develop a $(8+4\Delta)$-approximation algorithm.

The performance ratios are usually larger if jobs are released over time: In the offline setting with identical machines the best known approximation algorithm has performance ratio $2+\Delta$; see Schulz [19]. This performance ratio is also achieved for unrelated machines [22] and by a randomized online algorithm [19]. In the online setting there exist furthermore a deterministic $(\max\{2.618, 2.309+1.309\Delta\})$-approximation on identical machines [19] and a deterministic $(144+72\Delta)$-approximation on unrelated machines [8].

Our Contribution and Outline. We present the first progress on the approximability of the basic stochastic scheduling problem on identical parallel machines with expected total weighted completion time objective $P|\boldsymbol{p}_j \sim \text{stoch}|\text{E}[\sum w_j \boldsymbol{C}_j]$ since the seminal work of Möhring et al. [17]; see Figure 1. We prove that WSEPT achieves performance ratio

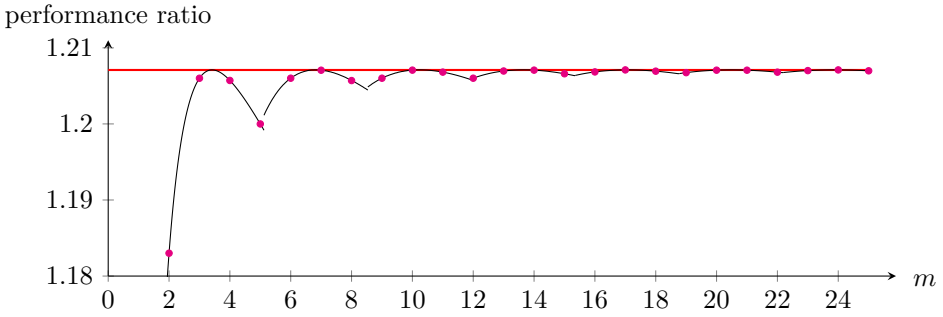


Figure 2 Graph of the function $m \mapsto 1 + \frac{1}{2}(\sqrt{(2m - k_m)k_m} - k_m)/m$, which for $m \in \mathbb{N}$ gives the worst-case approximation ratio of WSPT for $P||\sum w_j C_j$ with m machines (dots), compared to the machine-independent Kawaguchi-Kyan bound.

$$1 + \tfrac{1}{2} \min \left\{ \frac{\sqrt{(2m - k_m)k_m} - k_m}{m}, \frac{1}{1 + \min\{2, \sqrt{2 + 2\Delta}\}} \right\} (1 + \Delta), \tag{1}$$

where $k_m := \lfloor (1 - \frac{1}{2}\sqrt{2})m \rceil$ is the nearest integer to $(1 - \frac{1}{2}\sqrt{2})m$. Notice that, for every number of machines m, the performance ratio given by the first term of the minimum in (1) is bounded from above by $1 + \frac{1}{2}(\sqrt{2} - 1)(1 + \Delta)$, and for $m \to \infty$ it converges to this bound. As $(1 + \min\{2, \sqrt{2 + 2\Delta}\})^{-1} \leq \sqrt{2} - 1$ for all $\Delta > 0$, when considering an arbitrary number of machines, the second term in the minimum dominates the first term. In the following, we list several points that emphasize the significance of the new performance ratio (1).

- For the special case of deterministic scheduling (i.e., $\Delta = 0$), the machine-independent performance ratio in (1) matches the Kawaguchi-Kyan bound $\frac{1}{2}(1 + \sqrt{2})$, which is known to be tight [14]. In particular, we dissolve the somewhat annoying discontinuity of the best previously known bounds [14, 17] at $\Delta = 0$; see Figure 1.
- Again for deterministic jobs, our machine-dependent bound $1 + \frac{1}{2}(\sqrt{(2m - k_m)k_m} - k_m)/m$ is tight and slightly improves the 30 years old Kawaguchi-Kyan bound for every fixed number of machines m; see Figure 2.
- For exponentially distributed processing times ($\Delta = 1$), our results imply that WSEPT achieves performance ratio 4/3. This solves an open problem by Jagtenberg et al. [12], who give a lower bound of 1.243 on WSEPT's performance and ask for an improvement of the previously best known upper bound of $2 - 1/m$ due to Möhring et al. [17].
- WSEPT's performance bound due to Möhring et al. [17] also holds for the MinIncrease policy, introduced by Megow et al. [15], which is a fixed-assignment policy, i.e. it determines for each job beforehand on which machine it is processed. Our stronger bound, together with a lower bound in [22], shows that WSEPT beats every fixed-assignment policy.

The improved performance ratio in (1) is derived as follows. In Section 2 we present one of the key results of this paper (see Theorem 1 below): If WSPT has performance ratio $1 + \beta$ for some β, then WSEPT achieves performance ratio $1 + \beta(1 + \Delta)$ for the stochastic scheduling problem. For the Kawaguchi-Kyan bound $1 + \beta = \frac{1}{2}(1 + \sqrt{2})$, this yields performance ratio $1 + \frac{1}{2}(\sqrt{2} - 1)(1 + \Delta)$. It is also interesting to notice that the performance ratio of Möhring et al. [17] follows from this theorem by plugging in $1 + \beta = 3/2 - 1/(2m)$, which is WSPT's performance ratio obtained from the bound of Eastman, Even, and Isaacs [4]; see Kawaguchi and Kyan [14]. We generalize Theorem 1 to performance ratios w.r.t. the weighted sum of α-points as objective function, where the α-point of a job j is the point in time when it has been processed for exactly αp_j time units.

The theorems derived in Section 2 provide tools to carry over bounds for the WSPT rule to the WSEPT rule. The concrete performance ratio for the WSEPT rule obtained this way thus depends on good bounds for the WSPT rule. In Section 3 we derive performance ratios for WSPT w.r.t. the weighted sum of α-points objective. For $\alpha = \frac{1}{2}$ this performance ratio follows easily from a result by Avidor, Azar, and Sgall [1]. As a consequence we obtain performance ratio $1 + \frac{1}{6}\max\{2, 1+\Delta\}$ for WSEPT. By optimizing the choice of α, we finally obtain the performance ratio $1 + \frac{1}{2}(1 + \min\{2, \sqrt{2+2\Delta}\})^{-1}(1+\Delta)$. The various bounds derived in Sections 2 and 3 are illustrated in Figure 1. Finally, in Section 4 the analysis of Schwiegelshohn [20] for the WSPT rule is refined for every fixed number of machines m, entailing the machine-dependent bound for the WSEPT rule in (1).

Due to space constraints, some proofs are omitted in this extended abstract. They can be found in the full version of this paper [11].

2 Performance ratio of the WSEPT rule

Let $\Delta \geq \mathrm{Var}[\boldsymbol{p}_j]/\mathrm{E}[\boldsymbol{p}_j]^2$ for all $j \in \{1, \ldots, n\}$. In Theorems 1 and 3 we demonstrate how performance ratios for the WSPT rule for deterministic scheduling can be carried over to stochastic scheduling. Theorem 1 starts out from a performance ratio for WSPT with respect to the usual objective function: the weighted sum of completion times. In Theorem 3 this is generalized insofar as a performance ratio for WSPT with the weighted sum of α-points as objective function is taken as a basis. Only Theorem 1 is proven in this extended abstract.

▶ **Theorem 1.** *If the WSPT rule on m machines has performance ratio $1 + \beta_m$ for the problem $P||\sum w_j C_j$, then the WSEPT rule achieves performance ratio $1 + \beta_m(1+\Delta)$ for $P|\boldsymbol{p}_j \sim \mathrm{stoch}|\mathrm{E}[\sum w_j \boldsymbol{C}_j]$ on m machines.*

The reason why the bound for the WSPT rule does not directly carry over to the WSEPT rule is that under a specific realization of the processing times the schedule obtained by the WSEPT policy may differ from the WSPT schedule for this realization. Still, under every realization the WSEPT schedule is a list schedule. Hence, usually a bound that is valid for every list schedule is used: The objective value of a list schedule on m machines is at most $1/m$ times the objective value of the list schedule on a single machine plus $(m-1)/m$ times the weighted sum of processing times. This bound, holding because a list scheduling policy assigns each job to the currently least loaded machine, is applied realizationwise to obtain a corresponding bound on the expected values in stochastic scheduling (cf. [17, Lemma 4.1]), which is then compared to an LP-based lower bound on the expected total weighted completion time under an optimal scheduling policy.

In order to benefit from the precise bounds known for the WSPT rule nevertheless, we regard the following auxiliary stochastic scheduling problem: For each job, instead of its weight w_j, we are given a weight factor ρ_j. The actual weight of a job is ρ_j times its actual processing time, i.e., if a job takes longer, it also becomes more important. The goal is again to minimize the total weighted completion time. For the thus defined stochastic scheduling problem list scheduling in order of the ρ_j has the nice property that it creates a WSPT schedule in every realization. So, any performance ratio of the WSPT rule directly carries over to this list scheduling policy for the auxiliary scheduling problem. In the following proof of Theorem 1 we first compare the expected total weighted completion time of a WSEPT schedule for the original problem to the expected objective value of the schedule obtained by list scheduling in order of ρ_j for the auxiliary problem, then apply the performance ratio of the WSPT rule, and finally compare the expected total weighted completion time of an

optimal schedule for the auxiliary problem to the expected objective value of the schedule obtained by an optimal policy for the original problem. The transitions between the two problems lead to the additional factor $1 + \Delta$ in the performance ratio.

Proof. Consider an instance of $P|\boldsymbol{p}_j \sim \text{stoch}|\mathrm{E}[\sum w_j \boldsymbol{C}_j]$ consisting of n jobs and m machines, and let $\beta \coloneqq \beta_m$ and $\rho_j \coloneqq w_j/\mathrm{E}[\boldsymbol{p}_j]$ for $j \in \{1, \dots, n\}$. For every realization $\vec{p} = (p_1, \dots, p_n)$ of the processing times we consider the instance $I(\vec{p})$ of $P||\sum w_j C_j$ which consists of n jobs with processing times $p_1, \dots, p_n$ and weights $\rho_1 p_1, \dots, \rho_n p_n$, so that the jobs in this instance have Smith ratios $\rho_1, \dots, \rho_n$ under all possible realizations. Therefore, for every realization $\vec{p}$ the schedule obtained by the WSEPT policy is a WSPT schedule for $I(\vec{p})$. Let $C_j^{\mathrm{WSEPT}}(\vec{p})$ denote the completion time of job j in the schedule obtained by the WSEPT policy in the realization $\vec{p}$, let $C_j^*(I(\vec{p}))$ denote its completion time in an optimal schedule for $I(\vec{p})$, and let $C_j^{\Pi^*}(\vec{p})$ denote j's completion time in the schedule constructed by an optimal stochastic scheduling policy under the realization $\vec{p}$. For every realization $\vec{p}$ of the processing times, since the WSEPT schedule obeys the WSPT rule for $I(\vec{p})$, its objective value is bounded by

$$\sum_{j=1}^{n} (\rho_j p_j) C_j^{\mathrm{WSEPT}}(\vec{p}) \leq (1+\beta) \cdot \sum_{j=1}^{n} (\rho_j p_j) C_j^*(I(\vec{p})).$$

As the schedule obtained by an optimal stochastic scheduling policy is feasible for $I(\vec{p})$,

$$\sum_{j=1}^{n} (\rho_j p_j) C_j^*(I(\vec{p})) \leq \sum_{j=1}^{n} (\rho_j p_j) C_j^{\Pi^*}(\vec{p}).$$

By putting these two inequalities together and taking expectations, we get the inequality

$$\mathrm{E}\left[\sum_{j=1}^{n} \rho_j \boldsymbol{p}_j \boldsymbol{C}_j^{\mathrm{WSEPT}}\right] \leq (1+\beta) \cdot \mathrm{E}\left[\sum_{j=1}^{n} \rho_j \boldsymbol{p}_j \boldsymbol{C}_j^{\Pi^*}\right],$$

where $\boldsymbol{C}_j^{\mathrm{WSEPT}} = C_j^{\mathrm{WSEPT}}((\boldsymbol{p}_1, \dots, \boldsymbol{p}_n))$ and $\boldsymbol{C}_j^{\Pi^*} = C_j^{\Pi^*}((\boldsymbol{p}_1, \dots, \boldsymbol{p}_n))$. Using the latter inequality, we can bound the expected total weighted completion time of the WSEPT rule:

$$\begin{aligned}
&\mathrm{E}\left[\sum_{j=1}^{n} w_j \boldsymbol{C}_j^{\mathrm{WSEPT}}\right] = \sum_{j=1}^{n} \rho_j \mathrm{E}[\boldsymbol{p}_j]\mathrm{E}[\boldsymbol{C}_j^{\mathrm{WSEPT}}] \\
&\overset{(*)}{=} \sum_{j=1}^{n} \rho_j \mathrm{E}[\boldsymbol{p}_j \boldsymbol{C}_j^{\mathrm{WSEPT}}] - \sum_{j=1}^{n} \rho_j \mathrm{Var}[\boldsymbol{p}_j] = \mathrm{E}\left[\sum_{j=1}^{n} \rho_j \boldsymbol{p}_j \boldsymbol{C}_j^{\mathrm{WSEPT}}\right] - \sum_{j=1}^{n} \rho_j \mathrm{Var}[\boldsymbol{p}_j] \\
&\leq (1+\beta)\mathrm{E}\left[\sum_{j=1}^{n} \rho_j \boldsymbol{p}_j \boldsymbol{C}_j^{\Pi^*}\right] - \sum_{j=1}^{n} \rho_j \mathrm{Var}[\boldsymbol{p}_j] = (1+\beta)\sum_{j=1}^{n} \rho_j \mathrm{E}[\boldsymbol{p}_j \boldsymbol{C}_j^{\Pi^*}] - \sum_{j=1}^{n} \rho_j \mathrm{Var}[\boldsymbol{p}_j] \\
&\overset{(*)}{=} (1+\beta) \cdot \left(\sum_{j=1}^{n} \rho_j \mathrm{E}[\boldsymbol{p}_j]\mathrm{E}[\boldsymbol{C}_j^{\Pi^*}] + \sum_{j=1}^{n} \rho_j \mathrm{Var}[\boldsymbol{p}_j]\right) - \sum_{j=1}^{n} \rho_j \mathrm{Var}[\boldsymbol{p}_j] \\
&= (1+\beta) \cdot \sum_{j=1}^{n} w_j \mathrm{E}[\boldsymbol{C}_j^{\Pi^*}] + \beta \sum_{j=1}^{n} \rho_j \mathrm{Var}[\boldsymbol{p}_j] \leq (1+\beta) \cdot \sum_{j=1}^{n} w_j \mathrm{E}[\boldsymbol{C}_j^{\Pi^*}] + \Delta\beta \sum_{j=1}^{n} w_j \mathrm{E}[\boldsymbol{p}_j] \\
&\leq (1+\beta(1+\Delta)) \cdot \sum_{j=1}^{n} w_j \mathrm{E}[\boldsymbol{C}_j^{\Pi^*}] = (1+\beta(1+\Delta)) \cdot \mathrm{E}\left[\sum_{j=1}^{n} w_j \boldsymbol{C}_j^{\Pi^*}\right].
\end{aligned}$$

The equalities marked with $(*)$ hold because for any stochastic scheduling policy Π and all j

$$\mathrm{E}[\boldsymbol{p}_j \boldsymbol{C}_j^\Pi] = \mathrm{E}[\boldsymbol{p}_j \boldsymbol{S}_j^\Pi] + \mathrm{E}[\boldsymbol{p}_j^2] \ = \mathrm{E}[\boldsymbol{p}_j]\mathrm{E}[\boldsymbol{S}_j^\Pi] + \mathrm{E}[\boldsymbol{p}_j]^2 + \mathrm{Var}[\boldsymbol{p}_j] = \mathrm{E}[\boldsymbol{p}_j]\mathrm{E}[\boldsymbol{C}_j^\Pi] + \mathrm{Var}[\boldsymbol{p}_j],$$

where $\boldsymbol{S}_j^\Pi$ denotes the starting time of job j under policy Π. The independence of $\boldsymbol{p}_j$ and $\boldsymbol{S}_j^\Pi$ follows from the independence of the processing times and the non-anticipativity of policy Π, and the last inequality uses the fact that $\mathrm{E}[\boldsymbol{p}_j] \leq \mathrm{E}[\boldsymbol{C}_j^{\Pi^*}]$ for every job j. ◀

By plugging in the Kawaguchi-Kyan bound, we immediately get the following performance ratio (see Figure 1).

▶ **Corollary 2.** *The WSEPT rule has performance ratio* $1 + \frac{1}{2}(\sqrt{2} - 1) \cdot (1 + \Delta)$ *for the problem* $P|\boldsymbol{p}_j \sim \mathrm{stoch}|\mathrm{E}[\sum w_j \boldsymbol{C}_j]$.

For $\alpha \in (0, 1]$ the *α-point* $C_j^{\mathrm{S}}(\alpha)$ of a job j is the (first) point in time at which it has been processed for αp_j time units. Introduced by Hall, Shmoys, and Wein [9] in order to convert a preemptive schedule into a non-preemptive one, the concept of α-points is often used in the *design* of algorithms (see e.g. [5, 2, 6, 21]). In contrast, we use them in the definition of an alternative objective function in order to improve the *analysis* of the WSEPT rule.

We consider as objective function the weighted sum of α-points $\sum_{j=1}^n w_j C_j^{\mathrm{S}}(\alpha)$ for $\alpha \in (0, 1]$. This differs only by the constant $(1 - \alpha) \sum_{j=1}^n w_j p_j$ from the weighted sum of completion times. So as for optimal solutions the objective functions are equivalent. The same applies to the stochastic variant, where the two objectives differ by $(1-\alpha) \sum_{j=1}^n w_j \mathrm{E}[\boldsymbol{p}_j]$. We now generalize Theorem 1 to the (expected) weighted sum of α-points.

▶ **Theorem 3.** *If the WSPT rule has performance ratio* $1 + \beta$ *for the deterministic problem* $P||\sum w_j C_j(\alpha)$*, then the WSEPT rule has performance ratio* $1 + \beta(1 + \Delta)$ *for the problem* $P|\boldsymbol{p}_j \sim \mathrm{stoch}|\mathrm{E}[\sum w_j \boldsymbol{C}_j(\alpha)]$ *and* $1 + \beta \cdot \max\{1, \alpha(1 + \Delta)\}$ *for* $P|\boldsymbol{p}_j \sim \mathrm{stoch}|\mathrm{E}[\sum w_j \boldsymbol{C}_j]$.

The proof relies on the same idea as the proof of Theorem 1, namely to apply the bound for $P||\sum w_j C_j(\alpha)$ realizationwise to the auxiliary stochastic problem described above. Theorem 1 follows from Theorem 3 by plugging in $\alpha = 1$.

3 Performance ratios for WSPT with weighted sum of α-points objective

In this section we derive performance ratios for $P||\sum w_j C_j(\alpha)$. The two classical performance guarantees for $P||\sum w_j C_j$ by Eastman, Even, and Isaacs [4] and by Kawaguchi and Kyan [14] can both be generalized to this problem. While the Eastman-Even-Isaacs bound can be established for every $\alpha \in (0, 1]$, the Kawaguchi-Kyan bound carries over only for $\alpha \in [\frac{1}{2}, 1]$. In return, the generalized Kawaguchi-Kyan bound is better for these α.

For a problem instance I denote by $\mathcal{N}(I)$ its job set, by $C_j^{\mathrm{WSPT}}(\alpha)(I)$ the α-point of job j in the WSPT schedule for I, and by $C_j^*(\alpha)(I)$ the α-point of job j in some fixed ('the') optimal schedule for I. Hence $C_j^{\mathrm{WSPT}}(1)(I) = C_j^{\mathrm{WSPT}}(I)$ is the completion time of j in the WSPT schedule, and analogously for the optimal schedule. Furthermore, let $M_i^{\mathrm{WSPT}}(I)$ and $M_i^*(I)$ denote the load of the i-th machine and $M_{\min}^{\mathrm{WSPT}}(I)$ and $M_{\min}^*(I)$ denote the load of the least loaded machine, in the WSPT schedule and the optimal schedule for I, respectively. Moreover, let $\mathrm{WSPT}_\alpha(I)$ and $\mathrm{OPT}_\alpha(I)$ denote the weighted sum of α-points of the schedule obtained by the WSPT rule and of the optimal schedule, respectively. Finally, denote by $\lambda_\alpha(I) \coloneqq \mathrm{WSPT}_\alpha(I)/\mathrm{OPT}_\alpha(I)$ the approximation ratio of the WSPT rule for the instance I. We assume that if multiple jobs have the same ratio w_j/p_j, the WSPT rule processes them according to an arbitrary job order given as part of the input.

It is a well-known fact (see e.g. [20]) that for the weighted sum of completion times objective the worst case for the WSPT rule occurs if all jobs have the same Smith ratio w_j/p_j. This generalizes to the weighted sum of α-points objective.

▶ **Lemma 4.** *For every $\alpha \in [0,1]$ and every instance I of $P||\sum w_j C_j(\alpha)$ there is an instance I' of $P||\sum p_j C_j(\alpha)$ with the same number of machines and $\lambda_\alpha(I') \geq \lambda_\alpha(I)$.*

The proof proceeds in the same way as the proof of Schwiegelshohn [20]. For unit Smith ratio instances the WSPT rule is nothing but list scheduling according to an arbitrary given order. Restricting to them has the benefit that the objective value of a schedule S can be computed easily from its machine loads, namely

$$\sum_{j=1}^{n} p_j C_j^{\mathrm{S}}(\tfrac{1}{2}) = \frac{1}{2}\sum_{i=1}^{m} (M_i^{\mathrm{S}})^2. \tag{2}$$

This classical observation can for example be found in the paper of Eastman et al. [4].

For the sum of the squares of the machine loads as objective function Avidor, Azar, and Sgall [1] showed that WSPT has performance ratio 4/3. So this also holds for the weighted sum of $\frac{1}{2}$-points. By plugging it in into Theorem 3, we get the following corollary.

▶ **Corollary 5.** *The WSEPT rule has performance ratio $1+\frac{1}{6}\max\{2, 1+\Delta\}$ for the scheduling problem $P|\boldsymbol{p}_j \sim \mathrm{stoch}|\mathrm{E}[\sum w_j \boldsymbol{C}_j]$.*

Now we generalize the bound of Eastman, Even, and Isaacs [4].

▶ **Theorem 6** (Generalized Eastman-Even-Isaacs bound)**.** *For every $\alpha \in (0,1]$ the WSPT rule has performance ratio*

$$1 + \frac{m-1}{2\alpha m} \leq 1 + \frac{1}{2\alpha}$$

for the problem $P||\sum w_j C_j(\alpha)$.

▶ Remark. The generalized Eastman-Even-Isaacs bound does not lead to better performance ratios for the WSEPT rule for $P|\boldsymbol{p}_j \sim \mathrm{stoch}|\mathrm{E}[\sum w_j \boldsymbol{C}_j]$ than the bound of Möhring et al. [17], as plugging in $\beta = \frac{m-1}{2\alpha m}$ into Theorem 3 leads to a performance ratio of

$$1 + \frac{m-1}{2\alpha m} \cdot \max\{1, \alpha(1+\Delta)\} \geq 1 + \frac{1}{2}(1+\Delta)\Big(1 - \frac{1}{m}\Big).$$

So far, by choosing $\alpha = 1$ and $\alpha = \frac{1}{2}$ we have derived the two performance ratios for the WSEPT rule labeled by [Cor. 2] and [Cor. 5] in Figure 1. The proofs of Schwiegelshohn [20] and of Avidor et al. [1] of the underlying bounds for WSPT are quite similar. Both consist of a sequence of steps that reduce the set of instances to be examined. In every such reduction step it is shown that for any instance I of the currently considered set there is an instance I' in a smaller set for which the approximation ratio of WSPT is not better. This can be generalized to arbitrary $\alpha \in [\frac{1}{2}, 1]$. The resulting performance ratios for WSPT lead by means of Theorem 3 to a family of different performance ratios for the WSEPT rule. Note that the performance ratio of WSEPT following from the result of Avidor et al. for $\alpha = \frac{1}{2}$ has better behavior for large values of Δ, while the performance ratio following from Kawaguchi and Kyan's result for $\alpha = 1$ is better for small Δ. This behavior generalizes to $\alpha \in [\frac{1}{2}, 1]$: the smaller the underlying α, the better the ratio for large Δ but the worse the ratio for small Δ. Finally, we take for every $\Delta > 0$ the minimum of all the derived bounds.

▶ **Theorem 7** (Generalized Kawaguchi-Kyan bound). *For every $\alpha \in [\frac{1}{2}, 1]$ the WSPT rule has performance ratio*

$$1 + \frac{1}{2\alpha + \sqrt{8\alpha}}$$

for $P||\sum w_j C_j(\alpha)$, and this bound is tight.

Combining this bound with Theorem 3 yields for every $\alpha \in [\frac{1}{2}, 1]$ the performance ratio $1 + \frac{1}{2}\max\{1/(\alpha+\sqrt{2\alpha}), (1+\Delta)/(1+\sqrt{2/\alpha})\}$ of WSEPT for $P|\boldsymbol{p}_j \sim \text{stoch}|\text{E}[\sum w_j \boldsymbol{C}_j]$. This is minimized at $\alpha := 1/\min\{2, 1+\Delta\}$, yielding the following performance ratio (see Figure 1).

▶ **Corollary 8.** *For $P|\boldsymbol{p}_j \sim \text{stoch}|\text{E}[\sum w_j \boldsymbol{C}_j]$ the WSEPT rule has performance ratio*

$$1 + \frac{1}{2} \cdot \frac{1}{1 + \min\{2, \sqrt{2(1+\Delta)}\}} \cdot (1+\Delta).$$

Proof sketch of Theorem 7

The proof of Theorem 7 is analogous to the proof of Schwiegelshohn [20], consisting of a sequence of reduction lemmas. Let $\alpha \in [\frac{1}{2}, 1]$, assume that $p_1 \geq \cdots \geq p_n$, and let $\ell := \max\{j \in \{1, \ldots, m\} \mid p_j \geq \frac{1}{m-j+1}\sum_{j'=j}^{n} p_{j'}\}$. Then we call the ℓ jobs with largest processing times *long* jobs and denote the set of long jobs by $\mathcal{L}$.

▶ **Lemma 9.** *For every instance I of $P||\sum p_j C_j(\alpha)$ and every $\varepsilon > 0$ there is an instance $I' = I'(\varepsilon)$ of $P||\sum p_j C_j(\alpha)$ with the same number of machines such that $\lambda_\alpha(I') \geq \lambda_\alpha(I)$ and*

1. *$M_{\min}^{\text{WSPT}}(I') = 1$,*
2. *every job j with $S_j^{\text{WSPT}}(I') < M_{\min}^{\text{WSPT}}(I')$ fulfills $C_j^{\text{WSPT}}(I') \leq M_{\min}^{\text{WSPT}}(I')$ and $p'_j < \varepsilon$,*
3. *in the optimal schedule for I' every machine is used only by a single long job or has load $M_{\min}^{*}(I')$.*

Like in Schwiegelshohn's paper, the lemma is proven by scaling the instance and splitting all jobs with $S_j^{\text{WSPT}} < M_{\min}^{\text{WSPT}}$ until they satisfy the conditions. Note that the restriction to $\alpha \geq \frac{1}{2}$ is needed for this lemma because for smaller α splitting jobs increases the objective value and can thence reduce the performance ratio.

From now on, we focus on instances I that fulfill the requirements of Lemma 9 for some $0 < \varepsilon < M_{\min}^{\text{WSPT}}(I)$. For a subset $\mathcal{J} \subseteq \mathcal{N}$ of jobs we write $p(\mathcal{J}) := \sum_{j\in\mathcal{J}} p_j$. We call the jobs in $\mathcal{S} := \{j \in \{1, \ldots, n\} \mid S_j^{\text{WSPT}}(I) < M_{\min}^{\text{WSPT}}\}$ *short* jobs. This set is disjoint from $\mathcal{L}$ because all jobs in $\mathcal{S}$ have processing time $p_j < \varepsilon$, and all jobs in $\mathcal{L}$ have processing time $p_j \geq p_\ell \geq \frac{1}{m-\ell+1}\sum_{j'=\ell}^{n} p_{j'} \geq M_{\min}^{\text{WSPT}} > \varepsilon$. Finally, we call the jobs in $\mathcal{M} := \mathcal{N}(I) \setminus (\mathcal{S} \cup \mathcal{L})$ *medium* jobs. For an instance I of the type of Lemma 9, in the optimal schedule every machine that does not process a long job has load $M_{\min}^{*}(I) = p(\mathcal{M} \cup \mathcal{S})/(m - |\mathcal{L}|)$. We may assume that every machine processes at most one non-short job (see Figure 3).

▶ **Lemma 10.** *For every instance I of $P||\sum p_j C_j(\alpha)$ satisfying the conditions of Lemma 9 there is an instance I' with $\lambda_\alpha(I') \geq \lambda_\alpha(I)$ that still satisfies the conditions of Lemma 9 and has the additional property that the processing times of all non-short jobs are equal.*

The proof is an adapted version of the proof of Corollary 5 in the paper of Schwiegelshohn.

Since by Lemma 9 reducing ε can only increase the approximation ratio, the worst-case approximation ratio is approached in the limit $\varepsilon \to 0$, which we will subsequently further investigate. In the limit the sum of the squared processing times of the short jobs is negligible, wherefore the limits for $\varepsilon \to 0$ of the objective values of the WSPT schedule and the optimal

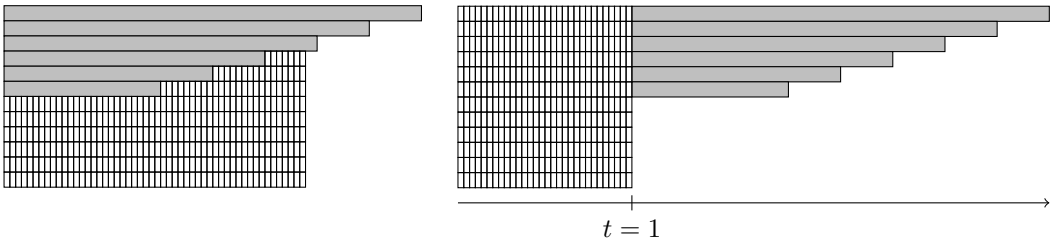


Figure 3 Optimal schedule and WSPT schedule for instance satisfying the conditions of Lemma 9.

schedule for an instance $I(\varepsilon)$ of the type of Lemma 10 only depend on two variables: the ratio s between the numbers of non-short jobs and machines and the duration x of the non-short jobs. The limit of the objective value of the WSPT schedule is given by

$$\lim_{\varepsilon \to 0} \mathrm{WSPT}_\alpha(I(\varepsilon)) = \frac{m}{2} + smx(1 + \alpha x).$$

For the optimal schedule the formula depends on whether the non-short jobs are medium or long. In the first case it is given by

$$\lim_{\varepsilon \to 0} \mathrm{OPT}_\alpha(I(\varepsilon)) = \frac{m}{2}(sx + 1)^2 + \left(\alpha - \frac{1}{2}\right) smx^2.$$

and in the second case by

$$\lim_{\varepsilon \to 0} \mathrm{OPT}_\alpha(I(\varepsilon)) = \alpha smx^2 + \frac{m}{2(1-s)}.$$

So we have to determine the maximum of the function

$$\lambda_{\mathrm{M}}(s,x) \coloneqq \frac{\frac{m}{2} + smx(1+\alpha x)}{\frac{m}{2}(sx+1)^2 + (\alpha - \frac{1}{2})smx^2} = \frac{2sx(\alpha x + 1) + 1}{s^2x^2 + sx((2\alpha - 1)x + 2) + 1}$$

on $\{(s,x) \mid 0 \le s < 1,\ 0 \le x \le 1/(1-s)\}$ and the maximum of

$$\lambda_{\mathrm{L}}(s,x) \coloneqq \frac{\frac{m}{2} + smx(1+\alpha x)}{\alpha smx^2 + \frac{m}{2(1-s)}} = \frac{(1-s)(2sx(\alpha x+1)+1)}{2\alpha s(1-s)x^2 + 1}$$

on the region $\{(s,x) \mid 0 \le s < 1,\ 1/(1-s) \le x\}$.

The partial derivative $\frac{\partial}{\partial x}\lambda_{\mathrm{M}}$ is positive on the feasible region, so for every fixed s the maximum of $\lambda_{\mathrm{M}}(s,\cdot)$ is attained at $x = \frac{1}{1-s}$, corresponding to the case that the non-short jobs are long. This case is also captured by the function λ_{L}.

For $x \to \infty$ the function λ_{L} converges to one. Hence, for every s the maximum of $\lambda_{\mathrm{L}}(s,\cdot)$ must be attained at a finite point x. The partial derivative $\frac{\partial}{\partial x}\lambda_{\mathrm{L}}$ has only one positive root, namely $x_s \coloneqq (\alpha s + \sqrt{(2(1-s)+\alpha s)\alpha s})/(2\alpha s(1-s)) > 1/(1-s)$. By plugging this in, we obtain $\lambda_{\mathrm{L}}(s,x_s) = 1 + \frac{1}{2}(\sqrt{(2(1-s)+\alpha s)\alpha s}/\alpha - s)$. The only root of the derivative of the function $s \mapsto \lambda_{\mathrm{L}}(s,x_s)$ that is less than 1 is $s \coloneqq 1/(2+\sqrt{2\alpha})$. Plugging this in yields the worst-case performance ratio

$$1 + \frac{1}{2\alpha + \sqrt{8\alpha}}.$$

Like the proofs of Kawaguchi and Kyan [14] and of Avidor et al. [1], this proof shows how the worst-case instances look like: They consist of short jobs of total length m and $1/(2+\sqrt{2\alpha})m$ long jobs of length $1 + \sqrt{2/\alpha}$. For $\alpha \in \{1/2, 1\}$ we recover the worst case instances of Avidor et al. and of Kawaguchi and Kyan.

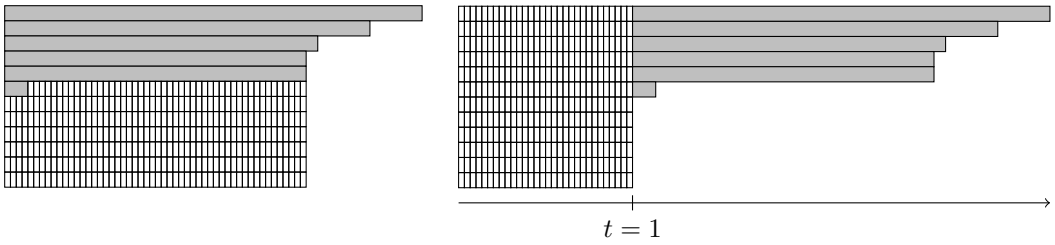


Figure 4 Optimal schedule and WSPT schedule for instance after the transformation of Lemma 12.

4 Performance ratio of the WSPT rule for a fixed number of machines

In this section we analyse the WSPT rule for the problem $P||\sum w_jC_j$ with a fixed number m of machines. The problem instances of Kawaguchi and Kyan [14] whose approximation ratios converge to $(1+\sqrt{2})/2$ consist of a set of infinitesimally short jobs with total processing time m, and a set of k jobs of length $1+\sqrt{2}$, where $k/m \to 1-\sqrt{2}/2$. Since $1-\sqrt{2}/2$ is irrational, the worst case ratio can only be approached if the number of machines goes to infinity. Rounding these instances for a fixed m by choosing k as the nearest integer to $\left(1-\frac{\sqrt{2}}{2}\right)m$ (in the following denoted by $\left\lfloor\left(1-\frac{\sqrt{2}}{2}\right)m\right\rceil$) yields at least a lower bound on the worst-case approximation ratio for $P||\sum w_jC_j$. As we will see, the worst-case instances for any fixed m look almost like that except that the length of the long jobs depends on m.

▶ **Theorem 11.** *For $P||\sum w_jC_j$ the WSPT rule has performance ratio*

$$1+\frac{1}{2}\frac{\sqrt{(2m-k_m)k_m}-k_m}{m}, \qquad \text{where} \quad k_m := \left\lfloor\left(1-\frac{\sqrt{2}}{2}\right)m\right\rceil.$$

Moreover, this bound is tight for every fixed $m \in \mathbb{N}$.

In the remainder we summarize the proof of this theorem. Lemmas 4 and 9 hold in particular for the weighted sum of completion times. Since the described transformations do not change the number of machines, also for a fixed number m of machines the worst case occurs in an instance of the form described in Lemma 9. However, we cannot apply Lemma 10 when m is fixed because the transformation in this lemma possibly changes the number of machines. As this is not allowed in our setting, we have to find different reductions. We first reduce to instances with at most one medium job and then reduce further to instances where all long jobs have equal length. Similar reductions are also carried out by Kalaitzis, Svensson, and Tarnawski [13].

▶ **Lemma 12.** *For every instance I of $P||\sum p_jC_j$ satisfying the conditions of Lemma 9 there is an instance I' with the same number of machines and $\lambda(I') \geq \lambda(I)$ that still satisfies the conditions of Lemma 9 and has the additional property that there is at most one medium job.*

For the instance I shown in Figure 3 the optimal and the WSPT schedule of the reduced instance I' are shown in Figure 4.

▶ **Lemma 13.** *For every instance I of $P||\sum p_jC_j$ satisfying the conditions of Lemma 12 there is an instance I' with the same number of machines and $\lambda(I') \geq \lambda(I)$ that still satisfies the conditions of Lemma 12 and additionally all long jobs have equal processing time.*

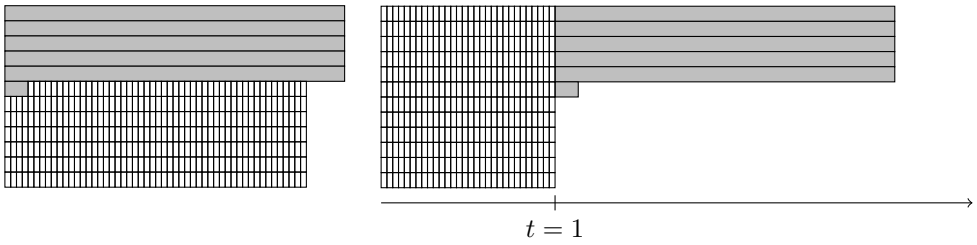


Figure 5 Optimal schedule and WSPT schedule for instance after the transformation of Lemma 13.

The reduction used in the proof of this lemma is illustrated in Figure 5.

As in Section 3 we will analyse the limit for $\varepsilon \to 0$. The limits of the objective values of the WSPT schedule and the optimal schedule for an instance $I(\varepsilon)$ of the type of Lemma 13 depend only on three variables: two real variables, namely the length x of the long jobs and the length y of the medium job ($y = 0$ if no medium job exists), and one integer variable: the number k of long jobs. They are given by

$$\lim_{\varepsilon \to 0} \mathrm{WSPT}(I(\varepsilon)) = \frac{m}{2} + kx(1+x) + y(1+y),$$
$$\lim_{\varepsilon \to 0} \mathrm{OPT}(I(\varepsilon)) = k \cdot x^2 + \frac{(m+y)^2}{2(m-k)} + \frac{y^2}{2}.$$

In Figure 6 of the full version [11] these formulas are illustrated via two-dimensional Gantt charts for the three different types of single-machine schedules used by the WSPT schedule and the optimal schedule, respectively. In order to describe a valid scheduling instance of the prescribed type, the values x, y, and k must lie in the domains

$$k \in \{0, \dots, m-1\}, \qquad y \begin{cases} \in \left[0, \frac{m}{m-k-1}\right] & \text{if } k < m-1, \\ = 0 & \text{if } k = m-1, \end{cases} \qquad x \in \left[\frac{y+m}{m-k}, \infty\right).$$

▶ **Lemma 14.** *The maximum of the ratio*

$$\lambda_m(x,y,k) \coloneqq \frac{\frac{m}{2} + kx(1+x) + y(1+y)}{k \cdot x^2 + \frac{(m+y)^2}{2(m-k)} + \frac{y^2}{2}} = \frac{(m-k)(2kx^2+2kx+2y^2+2y+m)}{(m-k)(2kx^2+y^2)+(y+m)^2}$$

on the feasible domains is $1 + \frac{1}{2}(\sqrt{(2m-k_m)k_m} - k_m)/m$*, and it is attained at*

$$k_m \coloneqq \left\lfloor \left(1 - \frac{1}{2}\sqrt{2}\right)m \right\rfloor, \qquad y_m \coloneqq 0, \qquad x_m \coloneqq \frac{m}{\sqrt{(2m-k_m)k_m} - k_m}.$$

The calculations leading to these values are similar to those in Section 3. This concludes the proof of Theorem 11.

In Figure 2 the function $m \mapsto \lambda_m(x_m, 0, k_m)$, whose values at integral m are exactly the worst case approximation ratios for instances with m machines, is depicted. The jumps and kinks occur when the number k_m of long jobs in the worst-case instance changes. By taking the limit for $m \to \infty$, we obtain alternative proof of the performance ratio $\frac{1}{2}(1+\sqrt{2})$ by Kawaguchi and Kyan [14], avoiding the somewhat complicated transformation and case distinction in the proof of Lemma 10 and Schwiegelshohn's proof [20]. For increasing m the tight performance ratio converges quite quickly to $\frac{1}{2}(1+\sqrt{2})$: the difference lies in $O(1/m^2)$. By plugging in the machine-dependent performance ratio into Theorem 1, we obtain the following performance ratio for the WSEPT rule.

▶ **Corollary 15.** *For instances with m machines of the problem $P|\boldsymbol{p}_j \sim \text{stoch}|\text{E}[\sum w_j \boldsymbol{C}_j]$ the WSEPT rule has performance ratio*

$$1 + \frac{1}{2} \cdot \frac{\sqrt{(2m - k_m)k_m} - k_m}{m} \cdot (1 + \Delta).$$

This bound is better than the bound of Corollary 8 only if m and Δ both are small. Even for two machines, it is outdone for large Δ.

References

1. Adi Avidor, Yossi Azar, and Jirí Sgall. Ancient and new algorithms for load balancing in the l_{p} norm. *Algorithmica*, 29(3):422–441, 2001. doi:10.1007/s004530010051.
2. Chandra Chekuri, Rajeev Motwani, Balas Natarajan, and Clifford Stein. Approximation techniques for average completion time scheduling. *SIAM Journal on Computing*, 31(1):146–166, 2001. doi:10.1137/s0097539797327180.
3. Wang Chi Cheung, Felix Fischer, Jannik Matuschke, and Nicole Megow. A $\Omega(\Delta^{1/2})$ gap example for the WSEPT policy. Cited as personal communication in Uetz: MDS Autumn School Approximation Algorithms for Stochastic Optimization, 2014. URL: http://www3.math.tu-berlin.de/MDS/summerschool14-material/Uetz-Exercises.pdf.
4. Willard L. Eastman, Shimon Even, and I. Martin Isaacs. Bounds for the optimal scheduling of n jobs on m processors. *Management Science*, 11(2):268–279, 1964. doi:10.1287/mnsc.11.2.268.
5. Michel X. Goemans. Improved approximation algorithms for scheduling with release dates. In Michael Saks, editor, *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 591–598. Society for Industrial and Applied Mathematics, 1997. URL: http://dl.acm.org/citation.cfm?id=314394.
6. Michel X. Goemans, Maurice Queyranne, Andreas S. Schulz, Martin Skutella, and Yaoguang Wang. Single machine scheduling with release dates. *SIAM J. Discrete Math.*, 15(2):165–192, 2002. doi:10.1137/S089548019936223X.
7. Ronald L. Graham, Eugene L. Lawler, Jan Karel Lenstra, and Alexander H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In Peter L. Hammer, Ellis L. Johnson, and Bernhard H. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979. doi:10.1016/S0167-5060(08)70356-X.
8. Varun Gupta, Benjamin Moseley, Marc Uetz, and Qiaomin Xie. Stochastic online scheduling on unrelated machines. In Friedrich Eisenbrand and Jochen Könemann, editors, *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, volume 10328 of *Lecture Notes in Computer Science*, pages 228–240. Springer, 2017. doi:10.1007/978-3-319-59250-3_19.
9. Leslie A. Hall, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In Éva Tardos, editor, *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 142–151, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=313852.313907.
10. Sungjin Im, Benjamin Moseley, and Kirk Pruhs. Stochastic scheduling of heavy-tailed jobs. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 474–486. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.474.
11. S. Jäger and M. Skutella. Generalizing the Kawaguchi-Kyan bound to stochastic parallel machine scheduling. *ArXiv e-prints*, jan 2018. arXiv:1801.01105.

12 Caroline Jagtenberg, Uwe Schwiegelshohn, and Marc Uetz. Analysis of smith's rule in stochastic machine scheduling. *Oper. Res. Lett.*, 41(6):570–575, 2013. doi:10.1016/j.orl.2013.08.001.

13 Christos Kalaitzis, Ola Svensson, and Jakub Tarnawski. Unrelated machine scheduling of jobs with uniform Smith ratios. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2654–2669, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics. arXiv:1607.07631.

14 Tsuyoshi Kawaguchi and Seiki Kyan. Worst case bound of an LRF schedule for the mean weighted flow-time problem. *SIAM J. Comput.*, 15(4):1119–1129, 1986. doi:10.1137/0215081.

15 Nicole Megow, Marc Uetz, and Tjark Vredeveld. Models and algorithms for stochastic online scheduling. *Math. Oper. Res.*, 31(3):513–525, 2006. doi:10.1287/moor.1060.0201.

16 Rolf H. Möhring, Franz Josef Radermacher, and Gideon Weiss. Stochastic scheduling problems I - general strategies. *Zeitschr. für OR*, 28(7):193–260, 1984. doi:10.1007/BF01919323.

17 Rolf H. Möhring, Andreas S. Schulz, and Marc Uetz. Approximation in stochastic scheduling: the power of lp-based priority policies. *J. ACM*, 46(6):924–942, 1999. doi:10.1145/331524.331530.

18 Michael H. Rothkopf. Scheduling with random service times. *Management Science*, 12(9):707–713, may 1966. doi:10.1287/mnsc.12.9.707.

19 Andreas S. Schulz. Stochastic online scheduling revisited. In Boting Yang, Ding-Zhu Du, and Cao An Wang, editors, *Combinatorial Optimization and Applications, Second International Conference, COCOA 2008, St. John's, NL, Canada, August 21-24, 2008. Proceedings*, volume 5165 of *Lecture Notes in Computer Science*, pages 448–457. Springer, 2008. doi:10.1007/978-3-540-85097-7_42.

20 Uwe Schwiegelshohn. An alternative proof of the kawaguchi-kyan bound for the largest-ratio-first rule. *Oper. Res. Lett.*, 39(4):255–259, 2011. doi:10.1016/j.orl.2011.06.007.

21 Martin Skutella. A 2.542-approximation for precedence constrained single machine scheduling with release dates and total weighted completion time objective. *Oper. Res. Lett.*, 44(5):676–679, 2016. doi:10.1016/j.orl.2016.07.016.

22 Martin Skutella, Maxim Sviridenko, and Marc Uetz. Unrelated machine scheduling with stochastic processing times. *Math. Oper. Res.*, 41(3):851–864, 2016. doi:10.1287/moor.2015.0757.

23 Martin Skutella and Gerhard J. Woeginger. A PTAS for minimizing the total weighted completion time on identical parallel machines. *Math. Oper. Res.*, 25(1):63–75, 2000. doi:10.1287/moor.25.1.63.15212.

24 Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, mar 1956. doi:10.1002/nav.3800030106.

25 Richard R. Weber, Pravin Varaiya, and Jean Walrand. Scheduling jobs with stochastically ordered processing times on parallel machines to minimize expected flowtime. *Journal of Applied Probability*, 23(3):841–847, sep 1986. doi:10.2307/3214023.

26 Gideon Weiss. Approximation results in parallel machnies stochastic scheduling. *Annals of Operations Research*, 26(1):195–242, 1990. doi:10.1007/BF02248591.

27 Gideon Weiss. Turnpike optimality of smith's rule in parallel machines stochastic scheduling. *Math. Oper. Res.*, 17(2):255–270, 1992. doi:10.1287/moor.17.2.255.

Space-Efficient Algorithms for Longest Increasing Subsequence

Masashi Kiyomi
Yokohama City University. Yokohama, Japan
masashi@yokohama-cu.ac.jp

Hirotaka Ono
Nagoya University. Nagoya, Japan
ono@nagoya-u.jp

Yota Otachi
Kumamoto University. Kumamoto, Japan
otachi@cs.kumamoto-u.ac.jp

Pascal Schweitzer
TU Kaiserslautern. Kaiserslautern, Germany
schweitzer@cs.uni-kl.de

Jun Tarui
The University of Electro-Communications. Chofu, Japan
tarui@ice.uec.ac.jp

Abstract

Given a sequence of integers, we want to find a longest increasing subsequence of the sequence. It is known that this problem can be solved in $O(n \log n)$ time and space. Our goal in this paper is to reduce the space consumption while keeping the time complexity small. For $\sqrt{n} \leq s \leq n$, we present algorithms that use $O(s \log n)$ bits and $O(\frac{1}{s} \cdot n^2 \cdot \log n)$ time for computing the length of a longest increasing subsequence, and $O(\frac{1}{s} \cdot n^2 \cdot \log^2 n)$ time for finding an actual subsequence. We also show that the time complexity of our algorithms is optimal up to polylogarithmic factors in the framework of sequential access algorithms with the prescribed amount of space.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Longest Increasing Subsequence, Patience Sorting, Space-efficient Algorithm

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.44

Related Version The full version is available at `http://arxiv.org/abs/1712.09230`.

Funding Partially supported by MEXT KAKENHI grant number 24106004.

1 Introduction

Given a sequence of integers (possibly with repetitions), the problem of finding a longest increasing subsequence (LIS, for short) is a classic problem in computer science which has many application areas including bioinfomatics and physics (see [38] and the references therein). It is known that LIS admits an $O(n \log n)$-time algorithm that uses $O(n \log n)$ bits of working space [37, 17, 2], where n is the length of the sequence.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 44; pp. 44:1–44:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A wide-spread algorithm achieving these bounds is Patience Sorting, devised by Mallows [24, 25, 26]. Given a sequence of length n, Patience Sorting partitions the elements of the sequence into so-called *piles*.

It can be shown that the number of piles coincides with the length of a longest increasing subsequence (see Section 3 for details). Combinatorial and statistical properties of the piles in Patience Sorting are well studied (see [2, 8, 33]).

However, with the dramatic increase of the typical data sizes in applications over the last decade, a main memory consumption in the order of $\Theta(n \log n)$ bits is excessive in many algorithmic contexts, especially for basic subroutines such as LIS. We therefore investigate the existence of space-efficient algorithms for LIS.

Our results. In this paper, we present the first space-efficient algorithms for LIS that are exact. We start by observing that when the input is restricted to permutations, an algorithm using $O(n)$ bits can be obtained straightforwardly by modifying a previously known algorithm (see Section 3.3). Next, we observe that a Savitch type algorithm [36] for this problem uses $O(\log^2 n)$ bits and thus runs in quasipolynomial time. However, we are mainly interested in space-efficient algorithms that also behave well with regard to running time. To this end we develop an algorithm that determines the length of a longest increasing subsequence using $O(\sqrt{n} \log n)$ bits which runs in $O(n^{1.5} \log n)$ time. Since the constants hidden in the O-notation are negligible, the algorithm, when executed in the main memory of a standard computer, may handle a peta-byte input on external storage.

More versatile, in fact, our space-efficient algorithm is *memory-adjustable* in the following sense. (See [3] for information on memory-adjustable algorithms.) When a memory bound s with $\sqrt{n} \leq s \leq n$ is given to the algorithm, it computes with $O(s \log n)$ bits of working space in $O(\frac{1}{s} \cdot n^2 \log n)$ time the length of a longest increasing subsequence. When $s = n$ our algorithm is equivalent to the previously known algorithms mentioned above. When $s = \sqrt{n}$ it uses, as claimed above, $O(\sqrt{n} \log n)$ bits and runs in $O(n^{1.5} \log n)$ time.

The algorithm only determines the length of a longest increasing subsequence. To actually find such a longest increasing subsequence, one can run the length-determining algorithm n times to successively construct the sought-after subsequence. This would give us a running time of $O(\frac{1}{s} \cdot n^3 \log n)$. However, we show that one can do much better, achieving a running time of $O(\frac{1}{s} \cdot n^2 \log^2 n)$ without any increase in space complexity, by recursively finding a *near-mid* element of a longest increasing subsequence.

To design the algorithms, we study the structure of the piles arising in Patience Sorting in depth and show that maintaining certain information regarding the piles suffices to simulate the algorithm. Roughly speaking, our algorithm divides the execution of Patience Sorting into $O(n/s)$ phases, and in each phase it computes in $O(n \log n)$ time information on the next $O(s)$ piles, while forgetting previous information.

Finally, we complement our algorithm with a lower bound in a restricted computational model. In the *sequential access model*, an algorithm can access the input only sequentially. We also consider further restricted algorithms in the *multi-pass model*, where an algorithm has to read the input sequentially from left to right and can repeat this multiple (not necessarily a constant number of) times. Our algorithm for the length works within the multi-pass model, while the one for finding a subsequence is a sequential access algorithm. Such algorithms are useful when large data is placed in an external storage that supports efficient sequential access. We show that the time complexity of our algorithms is optimal up to polylogarithmic factors in these models.

Related work. The problem of finding a longest increasing subsequence (LIS) is among the most basic algorithmic problems on integer arrays and has been studied continuously since the early 1960's. It is known that LIS can be solved in $O(n \log n)$ time and space [37, 17, 2], and that any comparison-based algorithm needs $\Omega(n \log n)$ comparisons even for computing the length of a longest increasing subsequence [17, 32]. For the special case of LIS where the input is restricted to permutations, there are $O(n \log \log n)$-time algorithms [20, 6, 12]. Patience Sorting, an efficient algorithm for LIS, has been a research topic in itself, especially in the context of Young tableaux [24, 25, 26, 2, 8, 33].

Recently, LIS has been studied intensively in the data-streaming model, where the input can be read only once (or a constant number of times) sequentially from left to right. This line of research was initiated by Liben-Nowell, Vee, and Zhu [22], who presented an exact one-pass algorithm and a lower bound for such algorithms. Their results were then improved and extended by many other groups [19, 38, 18, 34, 15, 28, 35]. These results give a deep understanding on streaming algorithms with a constant number of passes even under the settings with randomization and approximation. (For details on these models, see the very recent paper by Saks and Seshadhri [35] and the references therein.) On the other hand, multi-pass algorithms with a non-constant number of passes have not been studied for LIS.

While space-limited algorithms on both RAM and multi-pass models for basic problems have been studied since the early stage of algorithm theory, research in this field has recently intensified. Besides LIS, other frequently studied problems include sorting and selection [27, 7, 16, 30], graph searching [4, 14, 31, 9], geometric computation [10, 13, 5, 1], and k-SUM [39, 23].

2 Preliminaries

Let $\tau = \langle \tau(1), \tau(2), \ldots, \tau(n) \rangle$ be a sequence of n integers possibly with repetitions. For $1 \leq i_1 < \cdots < i_\ell \leq n$, the *subsequence* $\tau[i_1, \ldots, i_\ell]$ of τ is the sequence $\langle \tau(i_1), \ldots, \tau(i_\ell) \rangle$. A subsequence $\tau[i_1, \ldots, i_\ell]$ is an *increasing subsequence* of τ if $\tau(i_1) < \cdots < \tau(i_\ell)$. If $\tau(i_1) \leq \cdots \leq \tau(i_\ell)$, then the sequence τ is *non-decreasing*. We analogously define *decreasing subsequences* and *non-increasing subsequences*. By $\mathsf{lis}(\tau)$, we denote the length of a longest increasing subsequence of τ.

For example, consider a sequence $\tau_1 = \langle 2, 8, 4, 9, 5, 1, 7, 6, 3 \rangle$. It has an increasing subsequence $\tau_1[1, 3, 5, 8] = \langle 2, 4, 5, 6 \rangle$. Since there is no increasing subsequence of τ_1 with length 5 or more, we have $\mathsf{lis}(\tau_1) = 4$.

In the computational model in this paper, we use the RAM model with the following restrictions that are standard in the context of sublinear space algorithms. The input is in a read-only memory and the output must be produced on a write-only memory. We can use an additional memory that is readable and writable. Our goal is to minimize the size of the additional memory while keeping the running time fast. We measure space consumption in the number of bits used (instead of words) within the additional memory.

3 Patience Sorting

Since our algorithms are based on the classic Patience Sorting, we start by describing it in detail and recalling some important properties regarding its internal configurations.

Internally, the algorithm maintains a collection of piles. A *pile* is a stack of integers. It is equipped with the procedures push and top: the push procedure appends a new element to become the new top of the pile; and the top procedure simply returns the element on top of

Algorithm 1 PATIENCE SORTING.

```
1: set ℓ := 0 and initialize the dummy pile P_0 with the single element −∞
2: for i = 1 to n do
3:     if τ(i) > top(P_ℓ) then
4:         increment ℓ, let P_ℓ be a new empty pile, and set j := ℓ
5:     else
6:         set j to be the smallest index with τ(i) ≤ top(P_j)
7:     push τ(i) to P_j
8: return ℓ
```

the pile, which is always the one that was added last.

We describe how PATIENCE SORTING computes $\mathsf{lis}(\tau)$. See Algorithm 1. The algorithm scans the input τ from left to right (Line 2). It tries to push each newly read element $\tau(i)$ to a pile with a top element larger than or equal to $\tau(i)$. If on the one hand there is no such a pile, PATIENCE SORTING creates a new pile to which it pushes $\tau(i)$ (Line 4). On the other hand, if at least one such pile exists, PATIENCE SORTING pushes $\tau(i)$ to the oldest pile that satisfies the property (Line 6). After the scan, the number of piles is the output, which happens to be equal to $\mathsf{lis}(\tau)$ (Line 8).

We return to the sequence $\tau_1 = \langle 2, 8, 4, 9, 5, 1, 7, 6, 3 \rangle$ for an example. The following illustration shows the execution of Algorithm 1 on τ_1. In each step the bold number is the newly added element. The colored (and underlined) elements in the final piles form a longest increasing subsequence $\tau_1[1, 3, 5, 8] = \langle 2, 4, 5, 6 \rangle$, which can be extracted as described below.

				4			4				4	**5**		**1**	4	5		1	4	5		1	4	5	**6**				**3**			
2		2	**8**		2	8		2	8	**9**		2	8	9		2	8	9		2	8	9	**7**		2	8	9	7		2	8	9	7
P_1		P_1	P_2		P_1	P_2		P_1	P_2	P_3		P_1	P_2	P_3		P_1	P_2	P_3		P_1	P_2	P_3	P_4		P_1	P_2	P_3	P_4		P_1	P_2	P_3	P_4

▶ **Proposition 3.1** ([37, 17, 2]). *Given a sequence τ of length n,* PATIENCE SORTING *computes $\mathsf{lis}(\tau)$ in $O(n \log n)$ time using $O(n \log n)$ bits of working space.*

3.1 Correctness of Patience Sorting

It is observed in [8] that when the input is a permutation π, the elements of each pile form a decreasing subsequence of π. This observation easily generalizes as follows.

▶ **Observation 3.2.** *Given a sequence τ, the elements of each pile constructed by* PATIENCE SORTING *form a non-increasing subsequence of τ.*

Hence, any increasing subsequence of τ can contain at most one element in each pile. This implies that $\mathsf{lis}(\tau) \leq \ell$.

Now we show that $\mathsf{lis}(\tau) \geq \ell$. Using the piles, we can obtain an increasing subsequence of length ℓ, in reversed order, as follows [2]:

1. Pick an arbitrary element of P_ℓ;
2. For $1 \leq i < \ell$, let $\tau(h)$ be the element picked from P_{i+1}. Pick the element $\tau(h')$ that was the top element of P_i when $\tau(h)$ was pushed to P_{i+1}.

Since $h' < h$ and $\tau(h') < \tau(h)$ in each iteration, the ℓ elements that are selected form an increasing subsequence of τ. This completes the correctness proof for PATIENCE SORTING.

The proof above can be generalized to show the following characterization for the piles.

Algorithm 2 Computing $\mathsf{lis}(\pi)$ with $O(n)$ bits and in $O(n^2)$ time.

```
1: set ℓ := 0 and mark all elements in π as "unused"
2: while there is an "unused" element in π do
3:     increment ℓ and set t := ∞
4:     for i = 1 to n do                    ▷ this for-loop constructs the next pile implicitly
5:         if π(i) is unused and π(i) < t then
6:             mark π(i) as "used" and set t := π(i)        ▷ t is currently on top of P_ℓ
7: return ℓ
```

▶ **Proposition 3.3** ([8]). *$\tau(i) \in P_j$ if and only if a longest increasing subsequence of τ ending at $\tau(i)$ has length j.*

3.2 Time and space complexity of Patience Sorting

Observe that at any point in time, the top elements of the piles are ordered increasingly from left to right. Namely, $\mathtt{top}(P_k) < \mathtt{top}(P_{k'})$ if $k < k'$. This is observed in [8] for inputs with no repeated elements. We can see that the statement holds also for inputs with repetitions.

▶ **Observation 3.4.** *At any point in time during the execution of* Patience Sorting *and for any k and k' with $1 \le k < k' \le \ell$, we have $\mathtt{top}(P_k) < \mathtt{top}(P_{k'})$ if P_k and $P_{k'}$ are nonempty.*

The observation above implies that Line 6 of Algorithm 1 can be executed in $O(\log n)$ time by using binary search. Hence, Patience Sorting runs in $O(n \log n)$ time.

The total number of elements in the piles is $O(n)$ and thus Patience Sorting consumes $O(n \log n)$ bits. If it maintains all elements in the piles, it can compute an actual longest increasing subsequence in the same time and space complexity as described above. Note that to compute $\mathsf{lis}(\tau)$, it suffices to remember the top elements of the piles. However, the algorithm still uses $\Omega(n \log n)$ bits when $\mathsf{lis}(\tau) \in \Omega(n)$.

3.3 A simple $O(n)$-bits algorithm

Here we observe that, when the input is a permutation π of $\{1, \ldots, n\}$, $\mathsf{lis}(\pi)$ can be computed in $O(n^2)$ time with $O(n)$ bits of working space. The algorithm maintains a used/unused flag for each number in $\{1, \ldots, n\}$. Hence, this noncomparison-based algorithm cannot be generalized for general inputs directly.

Let τ be a sequence of integers without repetitions. A subsequence $\tau[i_1, \ldots, i_\ell]$ is the *left-to-right minima subsequence* if $\{i_1, \ldots, i_\ell\} = \{i : \tau(i) = \min\{\tau(j) : 1 \le j \le i\}\}$. In other words, the left-to-right minima subsequence is made by scanning τ from left to right and greedily picking elements to construct a maximal decreasing subsequence.

Burstein and Lankham [8, Lemma 2.9] showed that the first pile P_1 is the left-to-right minima subsequence of π and that the ith pile P_i is the left-to-right minima subsequence of a sequence obtained from π by removing all elements in the previous piles $P_1, \ldots, P_{i-1}$.

Algorithm 2 uses this characterization of piles. The correctness follows directly from the characterization. It uses a constant number of pointers of $O(\log n)$ bits and a Boolean table of length n for maintaining "used" and "unused" flags. Thus it uses $n + O(\log n)$ bits working space in total. The running time is $O(n^2)$: each for-loop takes $O(n)$ time and the loop is repeated at most n times.

4 An algorithm for computing the length

In this section, we present our main algorithm that computes $\mathsf{lis}(\tau)$ with $O(s \log n)$ bits in $O(\frac{1}{s} \cdot n^2 \log n)$ time for $\sqrt{n} \leq s \leq n$. Note that the algorithm here outputs the length $\mathsf{lis}(\tau)$ only. The next section discusses efficient solutions to actually compute a longest sequence.

In the following, by P_i for some i we mean the ith pile obtained by (completely) executing Patience Sorting unless otherwise stated. (We sometimes refer to a pile at some specific point of the execution.) Also, by $P_i(j)$ for $1 \leq j \leq |P_i|$ we denote the jth element added to P_i. That is, $P_i(1)$ is the first element added to P_i and $P_i(|P_i|)$ is the top element of P_i.

To avoid mixing up repeated elements, we assume that each element $\tau(j)$ of the piles is stored with its index j. In the following, we mean by "$\tau(j)$ is in P_i" that the jth element of τ is pushed to P_i. Also, by "$\tau(j)$ is $P_i(r)$" we mean that the jth element of τ is the rth element of P_i.

We start with an overview of our algorithm. It scans over the input $O(n/s)$ times. In each pass, it assumes that a pile P_i with at most s elements is given, which has been computed in the previous pass. Using this pile P_i, it filters out the elements in the previous piles $P_1, \ldots, P_{i-1}$. It then basically simulates Patience Sorting but only in order to compute the next $2s$ piles. As a result of the pass, it computes a new pile P_j with at most s elements such that $j \geq i + s$.

The following observation, that follows directly from the definition of Patience Sorting and Observation 3.4, will be useful for the purpose of filtering out elements in irrelevant piles.

▶ **Observation 4.1.** *Let $\tau(y) \in P_j$ with $j \neq i$. If $\tau(x)$ was the top element of P_i when $\tau(y)$ was pushed to P_j, then $j < i$ if $\tau(y) < \tau(x)$, and $j > i$ if $\tau(y) > \tau(x)$.*

Using Observation 4.1, we can obtain the following algorithmic lemma that plays an important role in the main algorithm.

▶ **Lemma 4.2.** *Having stored P_i explicitly in the additional memory and given an index $j > i$, the size $|P_k|$ for all $i+1 \leq k \leq \min\{j, \mathsf{lis}(\tau)\}$ can be computed in $O(n \log n)$ time with $O((|P_i| + j - i) \log n)$ bits. If $\mathsf{lis}(\tau) < j$, then we can compute $\mathsf{lis}(\tau)$ in the same time and space complexity.*

Proof. Recall that Patience Sorting scans the sequence τ from left to right and puts each element to the appropriate pile. We process the input in the same way except that we filter out, and thereby ignore, the elements in the piles P_h for which $h < i$ or $h > j$.

To this end, we use the following two filters whose correctness follows from Observation 4.1.

(Filtering P_h with $h < i$.) To filter out the elements that lie in P_h for some $h < i$, we maintain an index r that points to the element of P_i read most recently in the scan. Since P_i is given explicitly to the algorithm, we can maintain such a pointer r.

When we read a new element $\tau(x)$, we have three cases.

- If $\tau(x)$ is $P_i(r+1)$, then we increment the index r.
- Else if $\tau(x) < P_i(r)$, then $\tau(x)$ is ignored since it is in P_h for some $h < i$.
- Otherwise we have $\tau(x) > P_i(r)$. In this case $\tau(x)$ is in P_h for some $h > i$.

(Filtering P_h with $h > j$.) The elements in P_h for $h > j$ can be filtered without maintaining additional information as follows. Let again $\tau(x)$ be the newly read element.

- If no part of P_j has been constructed yet, then $\tau(x)$ is in P_h for some $h \leq j$.
- Otherwise, we compare $\tau(x)$ and the element $\tau(y)$ currently on the top of P_j.
  - If $\tau(x) > \tau(y)$, then $\tau(x)$ is in P_h for some $h > j$, and thus ignored.
  - Otherwise $\tau(x)$ is in P_h for some $h \leq j$.

Algorithm 3 Computing $|P_k|$ for all k with $i+1 \leq k \leq \min\{j, \mathsf{lis}(\tau)\}$ when P_i is given.

```
1: set r := 0                                   ▷ r points to the most recently read element in P_i
2: set ℓ := i                                   ▷ the largest index of the piles constructed so far
3: initialize p_{i+1}, ..., p_j to ∞            ▷ p_k is the element currently on top of P_k
4: initialize c_{i+1}, ..., c_j to 0            ▷ c_k is the current size of P_k
5: for x = 1 to n do
   ▷ filtering out irrelevant elements
6:     if τ(x) is P_i(r+1) then
7:         increment r and continue the for-loop
8:     else if τ(x) < P_i(r) or (ℓ ≥ j and τ(x) > p_j) then
9:         ignore the element and continue the for-loop
   ▷ push τ(x) to the appropriate pile
10:    if τ(x) > p_ℓ then
11:        increment ℓ and set h := ℓ
12:    else
13:        set h to be the smallest index with τ(i) < p_h
14:    set p_h := τ(x) and increment c_h
```

We simulate Patience Sorting only for the elements that pass both filters above. While doing so, we only maintain the top elements of the piles and additionally store the size of each pile. This requires at most $O((j-i)\log n)$ space, as required by the statement of the lemma. For details see Algorithm 3.

The running time remains the same since we only need constant number of additional steps for each step in Patience Sorting to filter out irrelevant elements. If P_j is still empty after this process, we can conclude that $\mathsf{lis}(\tau)$ is the index of the newest pile constructed. ◀

The proof of Lemma 4.2 can be easily adapted to also compute the pile P_j explicitly. For this, we simply additionally store all elements of P_j as they are added to the pile.

▶ **Lemma 4.3.** *Given P_i and an index j such that $i < j \leq \mathsf{lis}(\tau)$, we can compute P_j in $O(n \log n)$ time with $O((|P_i| + |P_j| + j - i)\log n)$ bits.*

Assembling the lemmas of this section, we now present our first main result. The corresponding pseudocode of the algorithm can be found in Algorithm 4.

▶ **Theorem 4.4.** *There is an algorithm that, given an integer s satisfying $\sqrt{n} \leq s \leq n$ and a sequence τ of length n, computes $\mathsf{lis}(\tau)$ in $O(\frac{1}{s} \cdot n^2 \log n)$ time with $O(s \log n)$ bits of space.*

Proof. To apply Lemmas 4.2 and 4.3 at the beginning, we start with a dummy pile P_0 with a single dummy entry $P_0(1) = -\infty$. In the following, assume that for some $i \geq 0$ we computed the pile P_i of size at most s explicitly. We repeat the following process until we find $\mathsf{lis}(\tau)$.

In each iteration, we first compute the size $|P_k|$ for $i+1 \leq k \leq i+2s$. During this process, we may find $\mathsf{lis}(\tau) < i+2s$. In such a case we output $\mathsf{lis}(\tau)$ and terminate. Otherwise, we find an index j such that $i+s+1 \leq j \leq i+2s$ and $|P_j| \leq n/s$. Since $s \geq \sqrt{n}$, it holds that $|P_j| \leq n/\sqrt{n} = \sqrt{n} \leq s$. We then compute P_j itself to replace i with j and repeat.

By Lemmas 4.2 and 4.3, each pass can be executed in $O(n \log n)$ time with $O(s \log n)$ bits. There are at most $\mathsf{lis}(\tau)/s$ iterations, since in each iteration the index i increases by at least s or $\mathsf{lis}(\tau)$ is determined. Since $\mathsf{lis}(\tau) \leq n$, the total running time is $O(\frac{1}{s} \cdot n^2 \log n)$. ◀

In the case of the smallest memory consumption we conclude the following corollary.

Algorithm 4 Computing $\mathsf{lis}(\tau)$ with $O(s \log n)$ bits in $O(\frac{1}{s} \cdot n^2 \log n)$ time.

```
1: set i := 0 and initialize the dummy pile P_0 with the single element −∞
2: loop
3:     compute the size of P_k for all k with i + 1 ≤ k ≤ i + 2s
4:     if we find lis(τ) < i + 2s then
5:         return lis(τ)
6:     let j be the largest index such that |P_j| ≤ s          ▷ i + s + 1 ≤ j ≤ i + 2s
7:     compute P_j and set i := j
```

▶ **Corollary 4.5.** *Given a sequence τ of length n, $\mathsf{lis}(\tau)$ can be computed in $O(n^{1.5} \log n)$ time with $O(\sqrt{n} \log n)$ bits of space.*

5 An algorithm for finding a longest increasing subsequence

It is easy to modify the algorithm in the previous section in such a way that it outputs an element of the final pile $P_{\mathsf{lis}(\tau)}$, which is the last element of a longest increasing subsequence by Proposition 3.3. Thus we can repeat the modified algorithm n times (considering only the elements smaller than and appearing before the last output) and actually find a longest increasing subsequence.[1] The running time of this naïve approach is $O(\frac{1}{s} \cdot n^3 \log n)$.

As we claimed before, we can do much better. In fact, we need only an additional multiplicative factor of $O(\log n)$ instead of $O(n)$ in the running time, while keeping the space complexity as it is. In the rest of this section, we prove the following theorem.

▶ **Theorem 5.1.** *There is an algorithm that, given an integer s satisfying $\sqrt{n} \leq s \leq n$ and a sequence τ of length n, computes a longest increasing subsequence of τ in $O(\frac{1}{s} \cdot n^2 \log^2 n)$ time using $O(s \log n)$ bits of space.*

▶ **Corollary 5.2.** *Given a sequence τ of length n, a longest increasing subsequence of τ can be found in $O(n^{1.5} \log^2 n)$ time with $O(\sqrt{n} \log n)$ bits of space.*

We should point out that the algorithm in this section is *not* a multi-pass algorithm. However, we can easily transform it without any increase in the time and space complexity so that it works as a sequential access algorithm.

5.1 High-level idea

We first find an element that is in a longest increasing subsequence roughly in the middle. As we will argue, this can be done in $O(\frac{1}{s} \cdot n^2 \log n)$ time with $O(s \log n)$ bits by running the algorithm from the previous section twice, once in the ordinary then once in the reversed way. We then divide the input into the left and right parts at a near-mid element and recurse.

The space complexity remains the same and the time complexity increases only by an $O(\log n)$ multiplicative factor. The depth of recursion is $O(\log n)$ and at each level of recursion the total running time is $O(\frac{1}{s} \cdot n^2 \log n)$. To remember the path to the current recursion, we need some additional space, but it is bounded by $O(\log^2 n)$ bits.

[1] This algorithm outputs a longest increasing subsequence in the reversed order. One can access the input in the reversed order and find a longest *decreasing* subsequence to avoid this issue.

Algorithm 5 Reverse Patience Sorting.

```
1: set ℓ := 0 and initialize the dummy pile Q_0 with the single element +∞
2: for i = n to 1 do
3:     if τ(i) < top(Q_ℓ) then
4:         increment ℓ, let Q_ℓ to be a new empty pile, and set j := ℓ
5:     else
6:         set j to be the smallest index with τ(i) > top(Q_j)
7:     push τ(i) to Q_j
8: return ℓ
```

5.2 A subroutine for short longest increasing sequences

We first solve the base case in which $\mathsf{lis}(\tau) \in O(n/s)$. In this case, we use the original Patience Sorting and repeat it $O(n/s)$ times. We present the following general form first.

▶ **Lemma 5.3.** *Let τ be a sequences of length n and $\mathsf{lis}(\tau) = k$. Then a longest increasing subsequence of τ can be found in $O(k \cdot n \log k)$ time with $O(k \log n)$ bits.*

Proof. Without changing the time and space complexity, we can modify the original Patience Sorting so that

- it maintains only the top elements of the piles;
- it ignores the elements larger than or equal to a given upper bound; and
- it outputs an element in the final pile.

We run the modified algorithm $\mathsf{lis}(\tau)$ times. In the first run, we have no upper bound. In the succeeding runs, we set the upper bound to be the output of the previous run. In each run the input to the algorithm is the initial part of the sequence that ends right before the last output. The entire output forms a longest increasing sequence of τ.[2]

Since $\mathsf{lis}(\tau) = k$, modified Patience Sorting maintains only k piles. Thus each run takes $O(n \log k)$ time and uses $O(k \log n)$ bits. The lemma follows since this is repeated k times and each round only stores $O(\log n)$ bits of information from the previous round. ◀

The following special form of the lemma above holds since $n/s \leq s$ when $s \geq \sqrt{n}$.

▶ **Corollary 5.4.** *Let τ be a sequence of length n and $\mathsf{lis}(\tau) \in O(n/s)$ for some s with $\sqrt{n} \leq s \leq n$. A longest increasing subsequence of τ can be found in $O(\frac{1}{s} \cdot n^2 \log n)$ time with $O(s \log n)$ bits.*

5.3 A key lemma

As mentioned above, we use a reversed version of our algorithm. Reverse Patience Sorting is the reversed version of Patience Sorting: it reads the input from right to left and uses the reversed inequalities. (See Algorithm 5.) Reverse Patience Sorting computes the length of a longest decreasing subsequence in the reversed sequence, which is a longest increasing subsequence in the original sequence. Since the difference between the two algorithms is small, we can easily modify our algorithm in Section 4 for the length so that it simulates Reverse Patience Sorting instead of Patience Sorting.

[2] Again this output is reversed. We can also compute the output in nonreversed order as discussed before.

Let Q_i be the ith pile constructed by Reverse Patience Sorting as in Algorithm 5. Using Proposition 3.3, we can show that for each $\tau(i)$ in Q_j, the longest decreasing subsequence of the reversal of τ ending at $\tau(i)$ has length j. This is equivalent to the following observation.

▶ **Observation 5.5.** *$\tau(i) \in Q_j$ if and only if a longest increasing subsequence of τ starting at $\tau(i)$ has length j.*

This observation immediately gives the key lemma below.

▶ **Lemma 5.6.** *$P_k \cap Q_{\mathsf{lis}(\tau)-k+1} \neq \emptyset$ for all k with $1 \le k \le \mathsf{lis}(\tau)$.*

Proof. Let $\langle \tau(i_1), \ldots, \tau(i_\ell) \rangle$ be a longest increasing subsequence of τ. Proposition 3.3 implies that $\tau(i_k) \in P_k$. The subsequence $\langle \tau(i_k), \ldots, \tau(i_\ell) \rangle$ is a longest increasing subsequence of τ starting at $\tau(i_k)$ since otherwise $\langle \tau(i_1), \ldots, \tau(i_\ell) \rangle$ is not longest. Since the length of $\langle \tau(i_k), \ldots, \tau(i_\ell) \rangle$ is $i_\ell - k + 1 = \mathsf{lis}(\tau) - k + 1$, we have $\tau(k) \in Q_{\mathsf{lis}(\tau)-k+1}$. ◀

Note that the elements of P_k and $Q_{\mathsf{lis}(\tau)-k+1}$ are not the same in general. For example, by applying Reverse Patience Sorting to $\tau_1 = \langle 2, 8, 4, 9, 5, 1, 7, 6, 3 \rangle$, we get $Q_1 = \langle 3, 6, 7, 9 \rangle$, $Q_2 = \langle 1, 5, 8 \rangle$, $Q_3 = \langle 4 \rangle$, and $Q_4 = \langle 2 \rangle$ as below. (Recall that $P_1 = \langle 2, 1 \rangle$, $P_2 = \langle 8, 4, 3 \rangle$, $P_3 = \langle 9, 5 \rangle$, and $P_4 = \langle 7, 6 \rangle$.) The following diagram depicts the situation. The elements shared by P_k and $Q_{\mathsf{lis}(\tau)-k+1}$ are colored and underlined.

								9		9			9			9															
		7		7			7			7			7	**8**		7	8						3							
	6		6		6	**5**		6	5		6	5		6	5		6	5				1	4	5	6						
3		3		**1**			3		1		3	1	**4**			4										2	8	9	7		
Q_1	Q_1	Q_1	Q_1	Q_2	Q_1	Q_2	Q_1	Q_2		Q_1	Q_2	Q_3	Q_1	Q_2	Q_3	Q_1	Q_2	Q_3	Q_4			P_1	P_2	P_3	P_4						

5.4 The algorithm

We first explain the subroutine for finding a near-mid element in a longest increasing subsequence.

▶ **Lemma 5.7.** *Let s be an integer satisfying $\sqrt{n} \le s \le n$. Given a sequence τ of length n, the kth element of a longest increasing subsequence of τ for some k with $\mathsf{lis}(\tau)/2 \le k < \mathsf{lis}(\tau)/2 + n/s$ can be found in $O(\frac{1}{s} \cdot n^2 \log n)$ time using $O(s \log n)$ bits of space.*

Proof. We slightly modify Algorithm 4 so that it finds an index k and outputs P_k such that $|P_k| \le s$ and $\mathsf{lis}(\tau)/2 \le k \le \mathsf{lis}(\tau)/2 + n/s$. Such a k exists since the average of $|P_i|$ for $\mathsf{lis}(\tau)/2 \le i < \mathsf{lis}(\tau)/2 + n/s$ is at most s. The time and space complexity of this phase are as required by the lemma.

We now find an element in $P_k \cap Q_{\mathsf{lis}(\tau)-k+1}$. Since the size $|Q_{\mathsf{lis}(\tau)-k+1}|$ is not bounded by $O(s)$ in general, we cannot store $Q_{\mathsf{lis}(\tau)-k+1}$ itself. Instead use the reversed version of the algorithm in Section 4 to enumerate it. Each time we find an element in $Q_{\mathsf{lis}(\tau)-k+1}$, we check whether it is included in P_k. This can be done with no loss in the running time since P_k is sorted and the elements of $Q_{\mathsf{lis}(\tau)-k+1}$ arrive in increasing order. ◀

The next technical but easy lemma allows us to split the input into two parts at an element of a longest increasing subsequence and to solve the smaller parts independently.

▶ **Lemma 5.8.** *Let $\tau(j)$ be the kth element of a longest increasing subsequence of a sequence τ. Let τ_L be the subsequence of $\tau[1, \ldots, j-1]$ formed by the elements smaller than $\tau(j)$. Similarly let τ_R be the subsequence of $\tau[j+1, \ldots, |\tau|]$ formed by the elements larger than $\tau(j)$. Then, a longest increasing subsequence of τ can be obtained by concatenating a longest increasing subsequence of τ_L, $\tau(j)$, and a longest increasing subsequence of τ_R, in this order.*

Algorithm 6 Recursively finding a longest increasing subsequence of ρ.

```
1:  RecursiveLIS(ρ, −∞, +∞)
2:  procedure RecursiveLIS(τ, lb, ub)
3:      τ′ := the subsequence of τ formed by the elements τ(i) such that lb < τ(i) < ub
            ▷ τ′ is not explicitly computed but provided by ignoring the irrelevant elements
4:      compute lis(τ′)
5:      if lis(τ′) ≤ 3|τ′|/s then
6:          output a longest increasing subsequence of τ′                    ▷ Lemma 5.3
7:      else
8:          find the kth element τ′(j) of a longest increasing subsequence of τ′
                for some k with lis(τ′)/2 ≤ k < lis(τ′)/2 + |τ′|/s
9:          RecursiveLIS(τ′[1, . . . , j − 1], lb, τ′(j))
10:         output τ′(j)
11:         RecursiveLIS(τ′[j + 1, . . . , |τ′|], τ′(j), ub)
```

Proof. Observe that the concatenated sequence is an increasing subsequence of τ. Thus it suffices to show that $\mathsf{lis}(\tau_L) + \mathsf{lis}(\tau_R) + 1 \geq \mathsf{lis}(\tau)$. Let $\tau[i_1, \ldots, i_{\mathsf{lis}(\tau)}]$ be a longest increasing subsequence of τ such that $i_k = j$. From the definition, $\tau[i_1, \ldots, i_{k-1}]$ is a subsequence of τ_L, and $\tau[i_{k+1}, \ldots, i_{\mathsf{lis}(\tau)}]$ is a subsequence of τ_R. Hence $\mathsf{lis}(\tau_L) \geq k - 1$ and $\mathsf{lis}(\tau_R) \geq \mathsf{lis}(\tau) - k$, and thus $\mathsf{lis}(\tau_L) + \mathsf{lis}(\tau_R) + 1 \geq \mathsf{lis}(\tau)$. ◀

As Lemma 5.8 suggests, after finding a near-mid element $\tau(k)$, we recurse into τ_L and τ_R. If the input τ' to a recursive call has small $\mathsf{lis}(\tau')$, we directly compute a longest increasing subsequence. See Algorithm 6 for details of the whole algorithm. Correctness follows from Lemma 5.8 and correctness of the subroutines.

5.5 Time and space complexity

In Theorem 5.1, the claimed running time is $O(\frac{1}{s} \cdot n^2 \log^2 n)$. To prove this, we first show that the depth of the recursion is $O(\log n)$. We then show that the total running time in each recursion level is $O(\frac{1}{s} \cdot n^2 \log n)$. The claimed running time is guaranteed by these bounds.

▶ **Lemma 5.9.** *Given a sequence τ, the depth of the recursions invoked by* RecursiveLIS *of Algorithm 6 is at most $\log_{6/5} \mathsf{lis}(\tau')$, where τ' is the subsequence of τ computed in Line 3.*

Proof. We proceed by induction on $\mathsf{lis}(\tau')$. If $\mathsf{lis}(\tau') \leq 3|\tau'|/s$, then no recursive call occurs, and hence the lemma holds. In the following, we assume that $\mathsf{lis}(\tau') = \ell > 3|\tau'|/s$ and that the statement of the lemma is true for any sequence τ'' with $\mathsf{lis}(\tau'') < \ell$.

Since $\ell > 3|\tau'|/s$, we recurse into two branches on subsequences of τ'. From the definition of k in Line 8 of Algorithm 6, the length of a longest increasing subsequence is less than $\ell/2 + |\tau'|/s$ in each branch. Since $\ell/2 + |\tau'|/s < \ell/2 + \ell/3 = 5\ell/6$, each branch invokes recursions of depth at most $\log_{6/5}(5\ell/6) = \log_{6/5} \ell - 1$. Therefore the maximum depth of the recursions invoked by their parent is at most $\log_{6/5} \ell$. ◀

▶ **Lemma 5.10.** *Given a sequence τ of length n, the total running time at each depth of recursion excluding further recursive calls in Algorithm 6 takes $O(\frac{1}{s} n^2 \log n)$ time.*

Proof. In one recursion level, we have many calls of RecursiveLIS on pairwise non-overlapping subsequences of τ. For each subsequence τ', the algorithm spends time $O(\frac{1}{s}|\tau'|^2 \log |\tau'|)$. Thus the total running time at a depth is $O(\sum_{\tau'} \frac{1}{s}|\tau'|^2 \log |\tau'|)$, which is $O(\frac{1}{s} n^2 \log n)$ since $\sum_{\tau'} |\tau'|^2 \leq |\tau|^2 = n^2$. ◀

Finally we consider the space complexity of Algorithm 6.

▶ **Lemma 5.11.** *Algorithm 6 uses $O(s \log n)$ bits of working space on sequences of length n.*

Proof. We have already shown that each subroutine uses $O(s \log n)$ bits. Moreover, this space of working memory can be discarded before another subroutine call occurs. Only a constant number of $O(\log n)$-bit words are passed to the new subroutine call. We additionally need to remember the stack trace of the recursion. The size of this additional information is bounded by $O(\log^2 n)$ bits since each recursive call is specified by a constant number of $O(\log n)$-bit words and the depth of recursion is $O(\log n)$ by Lemma 5.9. Since $\log^2 n \in O(s \log n)$ for $s \geq \sqrt{n}$, the lemma holds. ◀

6 Lower bound for algorithms with sequential access

An algorithm is a *sequential access* algorithm if it can access elements in the input array only sequentially. In our situation this means that for a given sequence, accessing the ith element of the sequence directly after having accessed the jth element of the sequence costs time at least linear in $|i - j|$. As opposed to the RAM, any Turing machine in which the input is given on single read-only tape has this property. Note that any lower bound for sequential access algorithms in an asymptotic form is applicable to multi-pass algorithms as well since every multi-pass algorithm can be simulated by a sequential access algorithm with the same asymptotic behavior. Although some of our algorithms are not multi-pass algorithms, it is straightforward to transform them to sequential access algorithms with the same time and space complexity.

To show a lower bound on the running time of sequential access algorithms with limited working space, we need the concept of communication complexity (see [21] for more details). Let f be a function. Given $\alpha \in \mathcal{A}$ to the first player Alice and $\beta \in \mathcal{B}$ to the second player Bob, the players want to compute $f(\alpha, \beta)$ together by sending bits to each other (possibly multiple times). The communication complexity of f is the maximum number of bits transmitted between Alice and Bob over all inputs by the best protocol for f. Now consider the following variant of the LIS problem: Alice gets the first half of a permutation π of $\{1, \ldots, 2n\}$ and Bob gets the second half. They compute $\mathsf{lis}(\pi)$ together. It is known that this problem has $\Omega(n)$ communication complexity (even with 2-sided error randomization) [22, 19, 38].

For sequential access algorithms, we can show the following lower bound by using the communication complexity lower bound mentioned above.

▶ **Theorem 6.1.** *Given a permutation π of $\{1, \ldots, 4n\}$, any sequential access (possibly randomized) algorithm computing $\mathsf{lis}(\pi)$ using b bits takes $\Omega(n^2/b)$ time.*

7 Concluding remarks

Our result raises the following question: "Do $o(\sqrt{n})$-space polynomial-time algorithms for LIS exist?" An unconditional 'no' answer would be surprising as it implies SC $\neq$ P $\cap$ PolyL, where SC (Steve's Class) is the class of problems that can be solved by an algorithm that simultaneously runs in polynomial-time and polylogarithmic-space [11, 29]. A possibly easier question asks for the existence of a log-space algorithm. For this question, one might be able to give some evidence for a 'no' answer by showing NL-hardness of (a decision version of) LIS.

As a final remark, we would like to mention some known results that have a mysterious coincidence in space complexity with our results. For $(1 + \epsilon)$-approximation of $\mathsf{lis}(\pi)$ by

one-pass streaming algorithms, it is known that $O(\sqrt{n/\epsilon} \cdot \log n)$ bits are sufficient [19] and $\Omega(\sqrt{n/\epsilon})$ bits are necessary [15, 18]. We were not able to find any connection here and do not claim anything concrete about this coincidence.

References

1 Hee-Kap Ahn, Nicola Baraldo, Eunjin Oh, and Francesco Silvestri. A time-space trade-off for triangulations of points in the plane. In Yixin Cao and Jianer Chen, editors, *Computing and Combinatorics - 23rd International Conference, COCOON 2017, Hong Kong, China, August 3-5, 2017, Proceedings*, volume 10392 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2017. doi:10.1007/978-3-319-62389-4_1.

2 David Aldous and Persi Diaconis. Longest increasing subsequences: from patience sorting to the Baik-Deift-Johansson theorem. *Bulletin of the American Mathematical Society*, 36(4):413–432, 1999. doi:10.1090/S0273-0979-99-00796-X.

3 Tetsuo Asano, Amr Elmasry, and Jyrki Katajainen. Priority queues and sorting for read-only data. In T.-H. Hubert Chan, Lap Chi Lau, and Luca Trevisan, editors, *Theory and Applications of Models of Computation, 10th International Conference, TAMC 2013, Hong Kong, China, May 20-22, 2013. Proceedings*, volume 7876 of *Lecture Notes in Computer Science*, pages 32–41. Springer, 2013. doi:10.1007/978-3-642-38236-9_4.

4 Tetsuo Asano, Taisuke Izumi, Masashi Kiyomi, Matsuo Konagaya, Hirotaka Ono, Yota Otachi, Pascal Schweitzer, Jun Tarui, and Ryuhei Uehara. Depth-first search using o(n) bits. In Hee-Kap Ahn and Chan-Su Shin, editors, *Algorithms and Computation - 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*, volume 8889 of *Lecture Notes in Computer Science*, pages 553–564. Springer, 2014. doi:10.1007/978-3-319-13075-0_44.

5 Bahareh Banyassady, Matias Korman, Wolfgang Mulzer, André van Renssen, Marcel Roeloffzen, Paul Seiferth, and Yannik Stein. Improved time-space trade-offs for computing voronoi diagrams. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 9:1–9:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.9.

6 Sergei Bespamyatnikh and Michael Segal. Enumerating longest increasing subsequences and patience sorting. *Inf. Process. Lett.*, 76(1-2):7–11, 2000. doi:10.1016/S0020-0190(00)00124-1.

7 Allan Borodin and Stephen A. Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM J. Comput.*, 11(2):287–297, 1982. doi:10.1137/0211022.

8 Alexander Burstein and Isaiah Lankham. Combinatorics of patience sorting piles. *Séminaire Lotharingien de Combinatoire*, 54A:B54Ab, 2006. URL: http://www.mat.univie.ac.at/~slc/wpapers/s54Aburlank.html.

9 Sankardeep Chakraborty and Srinivasa Rao Satti. Space-efficient algorithms for maximum cardinality search, stack bfs, queue BFS and applications. In Yixin Cao and Jianer Chen, editors, *Computing and Combinatorics - 23rd International Conference, COCOON 2017, Hong Kong, China, August 3-5, 2017, Proceedings*, volume 10392 of *Lecture Notes in Computer Science*, pages 87–98. Springer, 2017. doi:10.1007/978-3-319-62389-4_8.

10 Timothy M. Chan and Eric Y. Chen. Multi-pass geometric algorithms. *Discrete & Computational Geometry*, 37(1):79–102, 2007. doi:10.1007/s00454-006-1275-6.

11 Stephen A. Cook. Deterministic cfl's are accepted simultaneously in polynomial time and log squared space. In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 338–345. ACM, 1979. doi:10.1145/800135.804426.

12 Maxime Crochemore and Ely Porat. Fast computation of a longest increasing subsequence and application. *Inf. Comput.*, 208(9):1054–1059, 2010. doi:10.1016/j.ic.2010.04.003.

13 Omar Darwish and Amr Elmasry. Optimal time-space tradeoff for the 2d convex-hull problem. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 284–295. Springer, 2014. doi:10.1007/978-3-662-44777-2_24.

14 Amr Elmasry, Torben Hagerup, and Frank Kammer. Space-efficient basic graph algorithms. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 288–301. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.288.

15 Funda Ergün and Hossein Jowhari. On the monotonicity of a data stream. *Combinatorica*, 35(6):641–653, 2015. doi:10.1007/s00493-014-3035-1.

16 Greg N. Frederickson. Upper bounds for time-space trade-offs in sorting and selection. *J. Comput. Syst. Sci.*, 34(1):19–26, 1987. doi:10.1016/0022-0000(87)90002-X.

17 Michael L. Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1):29–35, 1975. doi:10.1016/0012-365X(75)90103-X.

18 Anna Gál and Parikshit Gopalan. Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. *SIAM J. Comput.*, 39(8):3463–3479, 2010. doi:10.1137/090770801.

19 Parikshit Gopalan, T.S. Jayram, Robert Krauthgamer, and Ravi Kumar. Estimating the sortedness of a data stream. In *SODA 2007*, pages 318–327, 2007. URL: http://dl.acm.org/citation.cfm?id=1283417.

20 James W. Hunt and Thomas G. Szymanski. A fast algorithm for computing longest subsequences. *Commun. ACM*, 20(5):350–353, 1977. doi:10.1145/359581.359603.

21 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997.

22 David Liben-Nowell, Erik Vee, and An Zhu. Finding longest increasing and common subsequences in streaming data. *J. Comb. Optim.*, 11(2):155–175, 2006. doi:10.1007/s10878-006-7125-x.

23 Andrea Lincoln, Virginia Vassilevska Williams, Joshua R. Wang, and R. Ryan Williams. Deterministic time-space trade-offs for k-sum. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 58:1–58:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.58.

24 C. L. Mallows. Problem 62-2, patience sorting. *SIAM Review*, 4(2):143–149, 1962. URL: http://www.jstor.org/stable/2028371.

25 C. L. Mallows. Problem 62-2. *SIAM Review*, 5(4):375–376, 1963. URL: http://www.jstor.org/stable/2028347.

26 C. L. Mallows. Patience sorting. *Bulletin of the Institute of Mathematics and its Applications*, 9:216–224, 1973.

27 J. Ian Munro and Mike Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980. doi:10.1016/0304-3975(80)90061-4.

28 Timothy Naumovitz and Michael E. Saks. A polylogarithmic space deterministic streaming algorithm for approximating distance to monotonicity. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1252–1262. SIAM, 2015. doi:10.1137/1.9781611973730.83.

29 Noam Nisan. RL $\subseteq$ SC. In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 619–623. ACM, 1992. doi:10.1145/129712.129772.

30 Jakob Pagter and Theis Rauhe. Optimal time-space trade-offs for sorting. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 264–268. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743455.

31 Michal Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 57:1–57:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.57.

32 Prakash Ramanan. Tight $\Omega(n \lg n)$ lower bound for finding a longest increasing subsequence. *International Journal of Computer Mathematics*, 65(3–4):161–164, 1997. doi:10.1080/00207169708804607.

33 Dan Romik. *The surprising mathematics of longest increasing subsequences*. Cambridge University Press, 2015. doi:10.1017/CBO9781139872003.

34 Michael E. Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1698–1709. SIAM, 2013. doi:10.1137/1.9781611973105.122.

35 Michael E. Saks and C. Seshadhri. Estimating the longest increasing sequence in polylogarithmic time. *SIAM J. Comput.*, 46(2):774–823, 2017. doi:10.1137/130942152.

36 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.

37 Craige Schensted. Longest increasing and decreasing subsequences. *Canadian Journal of Mathematics*, 13(2):179–191, 1961. doi:10.4153/CJM-1961-015-3.

38 Xiaoming Sun and David P. Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In *SODA 2007*, pages 336–345, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283419.

39 Joshua R. Wang. Space-efficient randomized algorithms for K-SUM. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 810–829. Springer, 2014. doi:10.1007/978-3-662-44777-2_67.

Rational, Recognizable, and Aperiodic Sets in the Partially Lossy Queue Monoid

Chris Köcher
Technische Universität Ilmenau, Automata and Logics Group
chris.koecher@tu-ilmenau.de

Abstract

Partially lossy queue monoids (or plq monoids) model the behavior of queues that can forget arbitrary parts of their content. While many decision problems on recognizable subsets in the plq monoid are decidable, most of them are undecidable if the sets are rational. In particular, in this monoid the classes of rational and recognizable subsets do not coincide. By restricting multiplication and iteration in the construction of rational sets and by allowing complementation we obtain precisely the class of recognizable sets. From these special rational expressions we can obtain an MSO logic describing the recognizable subsets. Moreover, we provide similar results for the class of aperiodic subsets in the plq monoid.

2012 ACM Subject Classification Theory of computation → Algebraic language theory, Theory of computation → Models of computation, Information systems → Data structures

Keywords and phrases Partially Lossy Queues, Transformation Monoid, Rational Sets, Recognizable Sets, Aperiodic Sets, MSO logic

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.45

Funding Supported by the DFG-Project "Speichermechanismen als Monoide", KU 1107/9-1.

1 Introduction

The study of different models of automata along with their expressiveness and algorithmic properties is one of the most important areas in automata theory. Many of these models differ in the mechanism to store their data, e.g., there are finite memories, pushdowns, (blind) counters, and infinite Turing tapes. Another very important mechanism is the so-called fifo queue (or channel), where data can be written to one end and read from the other end of its contents. If we equip these queues with a finite state automaton we obtain a Turing complete computation model [3], which results in the undecidability of all non-trivial decision problems on these devices. A surprising result was the decidability of some decision problems like reachability, fair termination or control-state-maintainability if the fifo queue is allowed to forget any part of its content at any time [8, 5, 1, 17].

To obtain some algebraic results on the behavior of these storage mechanisms we can model them as monoid of transformations. So, a single blind counter induces $(\mathbb{Z}, +)$ and a pushdown induces a polycyclic monoid [12]. Some important results on the transformation monoid of reliable queues can be found in [11]. Furthermore, in [14] we considered the transformation monoid of lossy queues. When studying the similarities and differences between those two monoids in [15] we found it convenient to join both, the reliable and lossy queues, respectively, into one model, the so-called *partially lossy queues* (or *plq*s). Those are given by their underlying alphabet A as well as a subset $U \subseteq A$ of letters that are unforgettable while the letters contained in $A \setminus U$ can be forgotten at any time. We denote the corresponding transformation monoid by $\mathcal{Q}(A, U)$ and call it the *partially lossy queue monoid* or *plq monoid*. Hence, with the help of plqs we can argue about reliable and lossy

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 45; pp. 45:1–45:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


queues at the same time, which results in the unification of some proofs considering these two models.

Another main topic in the theory of automata and formal languages is the study of regular languages. This revealed strong relations to logic, combinatorics, and algebra. For example, we can generalize the notion of regularity from free monoids to arbitrary monoids. This generalization results in two notions: the rational subsets, which are a generalization of languages that are described by regular expressions, and recognizable subsets, which are a generalization of sets accepted by finite automata (see, e.g., [2, 22]). Kleene's Theorem [13] states that both notions are equivalent in the free monoid.

In Section 3 we consider some algorithmic properties of rational subsets of the plq monoid. Such properties encountered increased attention in recent years, e.g., [16] provides a survey on the membership problem for rational sets. Since the rational sets in the polycyclic monoid (recall that this is the transformation monoid of a pushdown) are exactly the homomorphic images of a special subclass of the regular languages by [24], many decision problems like membership, intersection, universality, inclusion, and recognizability are decidable in this monoid. In this paper we will see that the membership problem of the plq monoid is NL-complete, but the other problems are undecidable, which we can prove by reduction from their counterparts in the direct product of $(\mathbb{N}, +)$ and $\{a, b\}^*$ (cf. [20, 9]).

If the given subsets are recognizable, all of the considered decision problems in plq monoids are decidable by known constructions from automata theory. Hence, the rational subsets are not effectively recognizable. Especially, we will see that the class of rational subsets in the plq monoid is not closed under intersection implying that the classes of rational and recognizable subsets do not coincide. In contrast, in polycyclic monoids the class of rational sets is closed under Boolean operations. However, the classes of rational and recognizable subsets do not coincide in these monoids since there are only two recognizable sets (the empty set and the monoid itself). But since there are even more recognizable sets in the plq monoid and since each recognizable subset is rational as well due to McKnight's Theorem [18], it is a natural question to ask in which cases a rational subset is recognizable.

For trace monoids, Ochmański could prove in [21] that it suffices to restrict the usage of the Kleene star in an appropriate way to characterize the recognizable subsets in the trace monoid. In Section 4 of this paper we will use an approach similar to Ochmański's to characterize the recognizable sets in terms of special rational sets in the plq monoid. Concretely, we will define some special restrictions on the usage of Kleene star and the concatenation to reach this target.

Another famous characterization of the regular languages is the definability in the monadic second-order logic MSO which was proven by Büchi in [4]. This result gave us an even brighter understanding than rational expressions of the formalization of the behavior of finite automata. Similar results about trace monoids can be found in [7, Chapter 10]. Hence, this motivates to find another MSO logic describing exactly the recognizable subsets in the plq monoid. In this paper we will give such a description.

The last result in this paper regards the connection between the aperiodic subsets, star-free subsets, and first-order logic. Recall that a set is aperiodic if it is accepted by a counter-free finite automaton and a set is star-free if it can be generated from finite sets by application of Boolean operations and concatenation, only. Schützenberger's Theorem [25] states that both classes coincide in the free monoid. This result gives a procedure to decide whether a given regular language is star-free. Additionally, in [10] it was proven that these classes also coincide in trace monoids. In contrast to these two cases this equality does not hold in the plq monoid. But we can characterize the aperiodic subsets in $\mathcal{Q}(A, U)$ with the help

of the same restrictions to star-freeness of subsets as in our result regarding the rational subsets. Finally, we prove similar to the results from [19, 7] that the aperiodic subsets in the plq monoid can be described by first-order formulas.

2 Preliminaries

At first, we need some basic definitions. So, let A be an alphabet. A word $v \in A^*$ is a *prefix* of $w \in A^*$ iff $w \in vA^*$. Similarly, v is a *suffix* of w iff $w \in A^*v$ and v is an *infix* of w iff $w \in A^*vA^*$. Furthermore, v is a *subword* of w (denoted by $v \preceq w$) iff there are $\ell \in \mathbb{N}$ and $a_1, \ldots, a_\ell \in A$ such that $v = a_1 \ldots a_\ell$ and $w \in A^*a_1A^*a_2 \ldots A^*a_\ell A^*$. Note that $\preceq$ is a partial ordering on A^*.

Let $S \subseteq A$. Then the *projection* $\pi_S \colon A^* \to S^*$ to S is the homomorphism induced by $\pi_S(a) = a$ for each $a \in S$ and $\pi_S(a) = \varepsilon$ for each $a \in A \setminus S$. Moreover, v is an *S-prefix* of w (denoted $v \leq_S w$) if there is a prefix w' of w such that $\pi_S(w') \preceq v \preceq w'$. In other words, we have $v \leq_S w$ if v is a subword of a prefix of w and contains all the letters from S in this prefix, e.g., we have $aa \leq_{\{a\}} abaab$ and $aa \not\leq_{\{b\}} abaab$. Note that $v \leq_\emptyset w$ means that v is a subword of w and $v \leq_A w$ means that v is a prefix of w.

2.1 Partially Lossy Queues

The partially lossy queue monoid (or plq monoid) models the behavior of a fifo-queue whose entries come from a finite set A. The unreliability of the queue stems from the fact that it can forget certain letters that we collect in the set $A \setminus U$. In other words, letters from $U \subseteq A$ are *non-forgettable* and those from $A \setminus U$ are *forgettable*.

So, let A be an alphabet of possible queue entries and let $U \subseteq A$ be the set of non-forgettable letters. The states of the queue are the words from A^*. Furthermore, we have some basic controllable actions on these queues: writing of a symbol $a \in A$ (denoted by a) and reading of $a \in A$ (denoted by $\overline{a}$). Thereby, we assume that the set $\overline{A}$ of all these reading operations $\overline{a}$ is a disjoint copy of A. So, $\Sigma := A \cup \overline{A}$ is the set of all controllable operations on the partially lossy queue. For a word $u = a_1 \ldots a_n \in A^*$ we write $\overline{u}$ for the word $\overline{a_1}\,\overline{a_2} \ldots \overline{a_n}$.

Formally, the action $a \in A$ appends the letter a to the state of the queue. The action $\overline{a} \in \overline{A}$ tries to cancel the letter a from the beginning of the current state of the queue. If this state does not start with a then the queue ends up in an error state. The lossiness of the queue is modeled by allowing it to forget arbitrary letters from $A \setminus U$ of its content at any moment.

Since a partially lossy queue with an underlying alphabet $A = \{a\}$ (independently of U) acts like a partially blind counter, the corresponding plq monoid is the bicyclic semigroup. On the first sight, the equality of these two transformation monoids seems to be counterintuitive. But it might be explained by the following observation: let $\mathcal{A}$ be an NFA equipped with one reliable counter. Then $\mathcal{A}$ accepts the same language as this NFA equipped with a lossy counter. Hence, from now on, we may exclude this case and assume $|A| \geq 2$.

Before defining the plq monoid we want to identify sequences of operations that have the same effect on any queue. In [15, Proposition 3.21] we proved that $u, v \in \Sigma^*$ act equally (denoted by $u \equiv v$) if, and only if, they can be transformed into each other by applying the equations from the following definition, only.

▶ **Definition 2.1.** Let $U \subseteq A$ be two finite sets. We define the binary relation $\equiv \subseteq (\Sigma^*)^2$ as the least congruence on Σ^* satisfying the following equations for $a, b \in A$, $c \in U$, and $w \in A^*$:

(a) $b\overline{a} \equiv \overline{a}b$ if $a \neq b$
(b) $a\overline{ab} \equiv \overline{a}a\overline{b}$
(c) $cwa\overline{a} \equiv cw\overline{a}a$
(d) $awa\overline{a} \equiv aw\overline{a}a$

Then the *partially lossy queue monoid* or *plq monoid* induced by (A, U) is the quotient $\mathcal{Q}(A, U) := \Sigma^*/_{\equiv}$. The natural epimorphism of $\equiv$ is $\eta\colon \Sigma^* \to \mathcal{Q}(A, U)\colon w \mapsto [w]$.

To handle the equivalence classes of $\equiv$ we want to define a normal form on this congruence. We do this by ordering the equations from Definition 2.1 from left to right, which results in an infinite semi-Thue system called $\mathcal{R}$.

Since the rules of $\mathcal{R}$ are length-preserving and move read actions to the left, it is terminating. Moreover, it is locally confluent by [15] and hence confluent. Therefore, for any word $u \in \Sigma^*$ there is a unique, irreducible word $\mathsf{nf}(u)$ with $u \to^* \mathsf{nf}(u)$, the so-called *normal form* of u.

▶ **Example 2.2.** Let $a, b \in A$ with $a \neq b$ and $q = aabb\overline{ab}$. If $a \notin U$ then we have

$$aabb\overline{ab} \to aab\overline{a}b\overline{b} \to aa\overline{a}bb\overline{b} \to a\overline{a}abb\overline{b} \to a\overline{a}ab\overline{b}b$$

and therefore $a\overline{a}ab\overline{b}b = \mathsf{nf}(aabb\overline{ab})$. Otherwise, i.e., if $a \in U$, we can apply Rule c to $a\overline{a}ab\overline{b}b$ and hence obtain $\mathsf{nf}(aabb\overline{ab}) = \overline{ab}aabb$.

From the definition of $\mathcal{R}$ we obtain that a word is in normal form if it starts with some read operations followed by a special shuffle of write and read operations where each read action $\overline{a}$ appears directly right from a. Thereby, the infixes $a\overline{a}$ in these words are divided by words from $(A \setminus (U \cup \{a\}))^*$, only. Formally, such shuffle of $u \in A^*$ and $\overline{v} \in \overline{A}^*$ is defined by $\langle\!\langle u, \overline{v} \rangle\!\rangle = w_1 a_1 \overline{a_1} w_2 a_2 \overline{a_2} \dots w_\ell a_\ell \overline{a_\ell} w_{\ell+1}$, where $v = a_1 \dots a_\ell$, $a_1, \dots, a_\ell \in A$, $u = a_1 w_1 \dots w_\ell a_\ell w_{\ell+1}$, and $w_i \in (A \setminus (U \cup \{a_i\}))^*$ for each $1 \leq i \leq \ell$. Then the set of all normal forms is

$$\mathsf{NF} = \{\overline{u}\langle\!\langle v, \overline{w} \rangle\!\rangle \mid u, v, w \in A^*, v \leq_U w\} = \overline{A}^* \left(\bigcup\nolimits_{a \in A} (A \setminus (U \cup \{a\}))^* a\overline{a} \right)^* A^* .$$

From this equation we can infer that $\mathsf{nf}(u) = \overline{u_1}\langle\!\langle u_2, \overline{u_3} \rangle\!\rangle$ is characterized by three components: The first component is the projection to the write actions $\pi(u) := u_2 = \pi_A(u)$ (note that the transitions of $\mathcal{R}$ preserve the relative ordering of the write operations). Similarly, the second is the projection to the read actions $\overline{\pi}(u) := u_1 u_3$ (note that we suppress the overlines in this projection). Finally, the third component is the *overlap* $\overline{\pi}_2(u) := u_3$ of u. Note that the characterization of NF from above implies that $\overline{\pi}_2(u) \leq_U \pi(u)$ holds. Additionally, we can define $\overline{\pi}_1(u) := u_1$.

▶ **Example 2.3.** Recall Example 2.2. There, in case of $a \notin U$ we have for $u = aabb\overline{ab}$: $\pi(u) = aabb$, $\overline{\pi}(u) = ab$, $\overline{\pi}_1(u) = \varepsilon$, and $\overline{\pi}_2(u) = ab$. Otherwise, if $a \in U$ we have $\overline{\pi}_1(u) = ab$ and $\overline{\pi}_2(u) = \varepsilon$.

While $\overline{\pi}_1(u)$ is defined using the semi-Thue system $\mathcal{R}$, it also has a natural meaning: $\overline{\pi}_1(u)$ is the shortest queue such that there is a run of the plq on execution of u that does not end up in the error state.

By [15, Proposition 3.21] the following holds about $\mathcal{R}$ and $\mathsf{nf}(u)$:

▶ **Proposition 2.4.** *Let $u, v \in \Sigma^*$. Then we have*

$$u \equiv v \iff \mathsf{nf}(u) = \mathsf{nf}(v) \iff (\pi(u), \overline{\pi}(u), \overline{\pi}_2(u)) = (\pi(v), \overline{\pi}(v), \overline{\pi}_2(v))\,.$$ ◀

With this main property in mind we can also apply π, $\overline{\pi}$, $\overline{\pi}_1$, and $\overline{\pi}_2$ to equivalence classes of $\equiv$ (i.e., elements from $\mathcal{Q}(A, U)$) instead of words from Σ^*.

Another question is the description of the normal form of $u\overline{v}$ for any $u, v \in A^*$. We have $\pi(u\overline{v}) = u$ and $\overline{\pi}(u\overline{v}) = v$. It remains to describe the overlap $\overline{\pi}_2(u\overline{v})$.

▶ **Lemma 2.5.** *Let $u, v \in A^*$. Then $\overline{\pi}_2(u\overline{v})$ is the longest suffix v' of v that satisfies $v' \leq_U u$.* ◀

Since $\equiv$ is a congruence we can infer $u \equiv \overline{\overline{\pi}_1(u)}\,\pi(u)\,\overline{\overline{\pi}_2(u)}$ for each $u \in \Sigma^*$ from Lemma 2.5.

2.2 Rationality, Recognizability, and Aperiodicity

Let $\mathcal{M}$ be a monoid. A subset $L \subseteq \mathcal{M}$ is called *rational* if it can be constructed from the finite subsets of $\mathcal{M}$ using union, concatenation, and Kleene iteration. The subset L is *recognizable* if there are a finite monoid $\mathcal{F}$ and a homomorphism $\phi\colon \mathcal{M} \to \mathcal{F}$ such that $L = \phi^{-1}(\phi(L))$, i.e., if L is accepted by an $\mathcal{M}$-automaton. It is well-known that the image of a rational set under a homomorphism is rational again and that the homomorphic preimage of a recognizable set also is recognizable. Furthermore, the class of recognizable subsets of $\mathcal{M}$ is closed under Boolean operations. Moreover, in a finitely generated monoid each recognizable set is rational by [18]. For example, this applies to $\mathcal{Q}(A, U)$ since this monoid is finitely generated. The converse direction is not true in general, e.g., in Theorem 3.4 we prove the existence of a rational subset of the plq monoid which is not recognizable. However, in free monoids generated by some alphabet Γ a subset $L \subseteq \Gamma^*$ is rational if, and only if, it is recognizable by Kleene's Theorem [13]. In this situation, we call L *regular*.

A recognizable set $L \subseteq \mathcal{M}$ is called *aperiodic* if there is $n \in \mathbb{N}$ such that for each $u, v, w \in \mathcal{M}$ we have $uv^n w \in L$ iff $uv^{n+1}w \in L$. It follows from [19] L is aperiodic if it is accepted by a counter-free $\mathcal{M}$-automaton. It is an easy exercise to prove that the class of aperiodic subsets is closed under Boolean operations and homomorphic preimages. By Schützenberger's Theorem [25] a language $L \subseteq \Gamma^*$ is aperiodic iff it is star-free. Recall that a set $L \subseteq \mathcal{M}$ is *star-free* if it can be constructed from finite subsets of $\mathcal{M}$ using union, concatenation, and complementation.

2.3 Logic and Languages

In this subsection we recall the logics on words and their correspondence to languages known from [26].

Let Γ be an alphabet. By FO we denote the set of first-order formulas built up from the atomic formulas of the form $x = y$, $x < y$, and $Q_a(x)$ for $a \in \Gamma$ where x and y are variables. To simplify notation we write $Q_S(x)$ instead of $\bigvee_{a \in S} Q_a(x)$ for any $S \subseteq \Gamma$.

Now let $w = a_1 \dots a_n \in \Gamma^*$. The *word model* for w is the relational structure $\underline{w} = (\mathsf{dom}(w), <^w, (Q_a^w)_{a \in \Gamma})$ where $\mathsf{dom}(w) = \{1, \dots, n\}$ is the set of letter positions of w, $<^w$ is the natural order on $\mathsf{dom}(w)$, and $Q_a^w = \{i \in \mathsf{dom}(w) \mid a_i = a\}$ is the set of positions of letters labeled with a. Then we write $\underline{w} \models \phi[p_1, \dots, p_n]$ for $p_1, \dots, p_n \in \mathsf{dom}(w)$ and a formula $\phi \in \mathsf{FO}$ (i.e., ϕ is satisfied in $\underline{w}$) if ϕ evaluates to true on interpretation of $=, <, Q_a$ as equality, $<^w$, and Q_a^w, respectively, and on interpretation of the free variables in ϕ as p_i's. Then the language defined by the sentence ϕ is $L(\phi) = \{w \in \Gamma^* \mid \underline{w} \models \phi\}$. We say that a language $L \subseteq \Gamma^*$ is FO-*definable* if there is $\phi \in \mathsf{FO}$ with $L = L(\phi)$.

By MSO (the *monadic second-order logic*) we denote the second-order extension of FO where the second-order variables are unary. Again, we say that $L \subseteq \Gamma^*$ is MSO-*definable* if there is $\phi \in \mathsf{MSO}$ with $L = L(\phi)$.

Büchi's Theorem [4] states that a language is regular iff it is MSO-definable. Moreover, a language is star-free and hence aperiodic iff it is FO-definable by [19].

3 Algorithmic Properties of Rational Subsets

This section studies decision problems concerning the rational subsets of $\mathcal{Q}(A, U)$. We will see that the classes of rational and recognizable subsets do not coincide. Especially, we prove that we cannot decide whether a given rational subset of the plq monoid is recognizable. Additionally, we prove that emptiness of intersection and the unique decipherability in $\mathcal{Q}(A, U)$ are undecidable. Though, we will see first, that the uniform membership problem is NL-complete.

So, let $w \in \Sigma^*$. Then one can show that the number of left-divisors of $[w]$ in $\mathcal{Q}(A, U)$ is at most $|w|^3$. Recall that in a monoid $\mathcal{M}$ u is a *left-divisor* of w if there is v such that $uv = w$. Hence, we can obtain a DFA with only $|w|^3$ many states that accepts $[w]$. In particular, similar to [11, Lemma 8.1] we can prove an even stronger result by using only logarithmic space on construction of this DFA. This implies the following theorem:

▶ **Theorem 3.1.** *Let A be an at least binary alphabet and $U \subseteq A$. Then the following rational subset membership problem for $\mathcal{Q}(A, U)$ is* NL*-complete: Given a word $w \in \Sigma^*$ and an NFA $\mathcal{A}$ over Σ. Is there a word $v \in L(\mathcal{A})$ with $w \equiv v$?*

Proof. Let $w \in \Sigma^*$ and let $\mathcal{A}$ be an NFA over Σ. Let $\mathcal{B}$ be the aforementioned DFA that can be constructed using only logarithmic additional space.

Then there exists $v \in L(\mathcal{A})$ with $w \equiv v$ if, and only if, $L(\mathcal{A}) \cap [w] \neq \emptyset$ if, and only if, $L(\mathcal{A}) \cap L(\mathcal{B}) \neq \emptyset$. Using an on-the-fly construction of $\mathcal{B}$, this can be decided nondeterministically in logarithmic space. Hence, the problem is in NL.

Since the free monoid A^* embeds into $\mathcal{Q}(A, U)$ and since the rational subset membership problem for A^* is NL-hard, we also get NL-hardness for $\mathcal{Q}(A, U)$. ◀

Now we will prove some negative algorithmic results on rational subsets of the plq monoid. In [11, Section 8] these undecidabilities for reliable queues could be inferred from an embedding of $\{a, b\}^* \times \{c, d\}^*$ into $\mathcal{Q}(A, A)$. Unfortunately, this does not work in arbitrary plq monoids since this direct product does not embed into $\mathcal{Q}(\{a, b\}, \emptyset)$ by [15, Theorem 6.14]. Though, we can prove all the undecidability results considered in [11] for any plq monoid.

Some of these results are based on an embedding of the monoid $\{a\}^* \times \{c, d\}^*$ into $\mathcal{Q}(A, U)$. Unfortunately, this does not help for the following two problems since their counterparts in $\{a\}^* \times \{c, d\}^*$ are decidable. Hence, we have to prove them directly.

The first considered decision problem is the unique decipherability problem in $\mathcal{Q}(A, U)$, i.e., the question whether a given finite set S freely generates S^*. To this end, we will use the undecidability of this problem in $\{a, b\}^* \times \{c, d\}^*$ by encoding the elements of the given set and adding another item.

▶ **Theorem 3.2.** *Let A be an at least binary alphabet and $U \subseteq A$. Then, given a finite set $S \subseteq \mathcal{Q}(A, U)$, it is undecidable whether S^* is freely generated by S.*

Proof. We prove this undecidability by reduction of this question for the monoid $\{a, b\}^* \times \{c, d\}^*$, which is undecidable by [6, Theorem 3.1]. So, let $a, b \in A$ be distinct letters and $S = \{(x_1, y_1), \ldots, (x_k, y_k)\}$. Define the embeddings $f\colon \{a, b\}^* \to A^*$ and $g\colon \{c, d\}^* \to A^*$ by

$f(a) = g(c) = aa$ and $f(b) = g(d) = ab$. Set $q_0 := [\overline{bbbb}]$, $q_i := [f(x_i)\overline{g(y_i)}]$ for any $1 \leq i \leq k$, and $T := \{q_i \mid 0 \leq i \leq k\} \subseteq \mathcal{Q}(A, U)$. Then we can show that S^* is freely generated by S iff T^* is freely generated by T. ◀

The next problem to consider is the emptiness of intersections of rational subsets in the plq monoid. Given two recognizable sets, this problem is decidable since the class of recognizable subsets is effectively closed under intersection. However, we will prove that this decidability does not hold for arbitrary rational subsets. As a corollary we can infer that the class of rational subsets is not effectively closed under intersection. Afterwards we will prove the existence of two rational subsets whose intersection is not rational. In consequence, the classes of rational and recognizable subsets do not coincide. Nevertheless, each recognizable set in $\mathcal{Q}(A, U)$ is rational due to [18] since the plq monoid is finitely generated.

▶ **Theorem 3.3.** *Let A be an at least binary alphabet and $U \subseteq A$. Then the emptiness of the intersection of two rational subsets of $\mathcal{Q}(A, U)$ is undecidable.*

Proof. We prove this by reduction of Post's Correspondence Problem (PCP), which is undecidable by [23]. So, let $a, b \in A$ be distinct letters and $I = ((x_1, y_1), \ldots, (x_k, y_k))$ be an instance of the PCP with $x_i, y_i \in A^*$. We define the following rational sets

$$X_I := \{p_i = [a^i b\overline{x_i}] \mid 1 \leq i \leq k\}^+ [\overline{a}][\overline{b}]^* \quad \text{and} \quad Y_I := \{q_i = [a^i b\overline{y_i}] \mid 1 \leq i \leq k\}^+ [\overline{a}][\overline{b}]^* .$$

We can show then that $X_I \cap Y_I \neq \emptyset$ if, and only if, I has a solution. ◀

To prove that the rational subsets are not closed under intersection and to prove the undecidability of the next problems we use an embedding of $\{a\}^* \times \{b, c\}^*$ into the plq monoid. Let $a, b \in A$ be distinct letters. Such an embedding is $\psi\colon \{a\}^* \times \{b, c\}^* \to \mathcal{Q}(A, U)$ with $\psi(a, \varepsilon) = [a]$, $\psi(\varepsilon, b) = [\overline{ab}]$, and $\psi(\varepsilon, c) = [\overline{abb}]$ by [15, Section 6.2].

▶ **Theorem 3.4.** *Let A be an at least binary alphabet and $U \subseteq A$. Then the set of rational subsets of $\mathcal{Q}(A, U)$ is not closed under intersection. In particular, there is a rational subset of $\mathcal{Q}(A, U)$ which is not recognizable.*

Proof. Consider the following rational relations:

$$R_1 = \{(a^m, b^m c^n) \mid m, n \in \mathbb{N}\} \quad \text{and} \quad R_2 = \{(a^m, b^n c^m) \mid m, n \in \mathbb{N}\} .$$

Then $\psi(R_1)$ and $\psi(R_2)$ are rational in $\mathcal{Q}(A, U)$. Suppose that $\psi(R_1) \cap \psi(R_2)$ is rational. Then there is a regular language $S \subseteq \Sigma^*$ with $\psi(R_1) \cap \psi(R_2) = \eta(S)$. Since ψ is injective we have $\psi(R_1) \cap \psi(R_2) = \psi(R_1 \cap R_2) = \psi(\{(a^n, b^n c^n) \mid n \in \mathbb{N}\})$. Hence, $\overline{\pi}(S) = \{(ab)^n (abb)^n \mid n \in \mathbb{N}\}$ would be regular since $\overline{\pi}$ is a homomorphism. But this is a contradiction to the Pumping Lemma. ◀

Gibbons and Rytter proved in [9] that universality and recognizability are undecidable in $\{a\}^* \times \{b, c\}^*$. Since ψ is an embedding of this monoid into the plq monoid, these undecidabilities imply the undecidability of their counterparts in the plq monoid.

▶ **Theorem 3.5.** *Let A be an at least binary alphabet and $U \subseteq A$. Then universality, inclusion, equality, and recognizability of rational subsets of $\mathcal{Q}(A, U)$ are undecidable.* ◀

4 Characterizations of the Recognizable Subsets

In Section 3 we have shown many decision problems on rational subsets of the plq monoid to be undecidable. We know that all these problems are decidable if the given subsets are recognizable from the known constructions in automata theory. Here, we want to give characterizations of the recognizable subsets in the manner of Kleene's and Büchi's Theorem [13, 4], i.e., we characterize the recognizable sets as certain rational sets and by logical means. At first, we state the main theorem. Later in this section we give the definitions of q-rational subsets and MSO_q and prove the correctness of this theorem.

▶ **Theorem 4.1** (Main Theorem). *Let A be an at least binary alphabet, $U \subseteq A$, and $S \subseteq \mathcal{Q}(A,U)$. Then the following are equivalent:*
(A) *S is recognizable.*
(B) *S is q-rational.*
(C) *S is MSO_q-definable.*

4.1 Some Helping Characterizations

Before we prove Theorem 4.1 we state two further characterizations which turned out to be convenient for simplification of the proof of Theorem 4.1. We know these characterizations from [11] for the recognizable subsets in the reliable queue monoid $\mathcal{Q}(A,A)$ and generalize them to plq monoids $\mathcal{Q}(A,U)$ with arbitrary subsets $U \subseteq A$. On the one hand, we prove the correspondence of recognizability in the plq monoid to regularity in the underlying free monoid. On the other hand, we show that each recognizable subset is a Boolean combination of sets $\pi^{-1}(R)$, $\overline{\pi}^{-1}(R)$ where $R \subseteq A^*$ is regular and some special sets Ω_ℓ for any $\ell \in \mathbb{N}$:

▶ **Definition 4.2.** Let $q \in \mathcal{Q}(A,U)$. Then the *overlap's bounded width* of q is

$$\omega(q) := \inf\{|\overline{\pi}_2(p)| \,:\, p \in \mathcal{Q}(A,U), \pi(p) = \pi(q), \overline{\pi}(p) = \overline{\pi}(q), |\overline{\pi}_2(q)| < |\overline{\pi}_2(p)|\} \,.$$

Furthermore, for $\ell \in \mathbb{N}$ set $\Omega_\ell := \{q \in \mathcal{Q}(A,U) \,|\, \omega(q) > \ell\}$.

The overlap's bounded width specifies the minimal length of the overlap of a word with the same projections having a longer overlap. If such word does not exist then we set this value to ∞.

▶ **Example 4.3.** Let $A = U = \{a,b\}$ and $q = \overline{aba}ba\bar{a}b\bar{b}abab$. Then there are two words with the same projections and longer overlaps: $q_1 = \overline{ab}a\bar{a}b\bar{b}a\bar{a}b\bar{b}ab$ and $q_2 = a\bar{a}b\bar{b}a\bar{a}b\bar{b}a\bar{a}b\bar{b}$. We have $|\overline{\pi}_2(q_1)| = 4$ and $|\overline{\pi}_2(q_2)| = 6$. Therefore, we have $\omega(q) = 4$, $\omega(q_1) = 6$, and $\omega(q_2) = \infty$. Hence, $q \in \Omega_3 \setminus \Omega_4$ holds.

From [11, Observation 9.1] we know that any non-trivial property of the overlap's width $|\overline{\pi}_2(q)|$ is not recognizable in $\mathcal{Q}(A,A)$. An appropriate alternative for the generators of the Boolean algebra of recognizable subsets was found in such kind of "overapproximation" of the overlap's length (note that $\omega(q) > |\overline{\pi}_2(q)|$ holds). Additionally, the following observations provide some more motivation of this notion:

▶ **Observation 4.4.** *Every $q \in \mathcal{Q}(A,U)$ is completely described by $\pi(q)$, $\overline{\pi}(q)$, and $\omega(q)$.* ◀

▶ **Observation 4.5.** *Let $\ell \in \mathbb{N}$ and $w \in \Sigma^*$. Then $\omega([w]) \leq \ell$ if, and only if, there is $u \in A^{\leq \ell}$ with $\overline{\pi}(w) \in A^*u$ and $u \leq_U \pi(w)$ such that $|\overline{\pi}_2(w)| < |u|$.* ◀

Now we can state the following equivalences which can be proven similar to [11, Theorem 9.4].

▶ **Theorem 4.6.** *Let A be an at least binary alphabet, $U \subseteq A$, and $S \subseteq \mathcal{Q}(A,U)$. Then the following are equivalent:*

1. *S is recognizable.*
2. *$\eta^{-1}(S) \cap \overline{A}^* A^* \overline{A}^*$ is regular.*
3. *S is a Boolean combination of sets of the form $\pi^{-1}(R)$ or $\overline{\pi}^{-1}(R)$ for some regular $R \subseteq A^*$ and the sets Ω_ℓ for some $\ell \in \mathbb{N}$.* ◀

4.2 From Recognizability to Q-Rational Subsets

In this subsection we prove that each recognizable subset in the plq monoid is q-rational. To this end, we first need to define this notion which is a restriction to the rational expressions. We need this restriction since we cannot translate Kleene's Theorem [13] to plq monoids due to Theorem 3.4. Though, we can use Ochmański's approach from [21] to generate the recognizable subsets. Concretely, we restrict the Kleene star and the concatenation of the plq monoid in an appropriate way. We call the sets generated by those operations *q-rational* and prove that these are exactly the recognizable subsets in the plq monoid.

At first, we prove that the class of recognizable subsets is not closed under iteration:

▶ Remark. Let $S = \{[a\overline{a}]\}$, which is trivially recognizable. Then $\eta^{-1}(S^*) \cap A^*\overline{A}^* \subseteq \Sigma^*$ is the set of all words $a^n\overline{a}^n$ with $n \in \mathbb{N}$ by Rule d in Definition 2.1. This language is not regular. Hence, $\eta^{-1}(S^*)$ is also not regular and therefore S^* is not recognizable.

This is a very similar situation as in trace monoids. Here, Ochmański proved in [21] that it suffices to restrict iteration to obtain some kind of rational expressions that are generating all the recognizable subsets [21]. Unfortunately, the class of recognizable subsets in the plq monoid also is not closed under product.

▶ Remark. Let $S = \{[a]\}^*$ and $T = \{[\overline{a}]\}^*$, which are recognizable. Then $\eta^{-1}(S \cdot T) \cap \overline{A}^* A^* \overline{A}^* \subseteq \Sigma^*$ is the set of all words $\overline{u_1}u_2\overline{u_3}$ with $u_1, u_2, u_3 \in a^*$ and $u_1 = \varepsilon$ or $|u_2| \leq |u_3|$ by Rule d in Definition 2.1. Since this language is not regular, $S \cdot T$ is not recognizable.

Hence, we have to restrict the use of the monoid's product in the construction of the so-called *q-rational* subsets. Next, we will define these subsets and afterwards we prove that these are a suitable restriction of rationality to describe exactly the recognizable subsets. But at first, we say that a subset of $\mathcal{Q}(A,U)$ is *q^+-rational* if it can be obtained by the following rules:

(1^+) $\pi^{-1}(\varepsilon)$, $\pi^{-1}(\emptyset) = \emptyset$, and $\pi^{-1}(a)$ for any $a \in A$ are q$^+$-rational
(2^+) if $S_1, S_2 \subseteq \mathcal{Q}(A,U)$ are q$^+$-rational then $S_1 \cup S_2$, $S_1 \cdot S_2$, and S_1^* are q$^+$-rational

Similarly, by replacing π^{-1} by $\overline{\pi}^{-1}$ in the rules above, we define the class of *q^--rational* subsets of $\mathcal{Q}(A,U)$.

▶ **Observation 4.7.** *Let $S \subseteq \mathcal{Q}(A,U)$. Then S is q^+-rational (q^--rational) if, and only if, there is some regular $R \subseteq A^*$ with $S = \pi^{-1}(R)$ ($S = \overline{\pi}^{-1}(R)$, resp.).* ◀

Finally, a subset of $\mathcal{Q}(A,U)$ is *q-rational* if it can be constructed from the following rules:

(1) if $S_1 \subseteq \mathcal{Q}(A,U)$ is q$^+$- or q$^-$-rational it also is q-rational
(2) if $S_1, S_2 \subseteq \mathcal{Q}(A,U)$ are q-rational then $S_1 \cup S_2$ and $\mathcal{Q}(A,U) \setminus S_1$ are q-rational
(3) if $S_1 \subseteq \mathcal{Q}(A,U)$ is q$^+$-rational and $S_2 \subseteq \mathcal{Q}(A,U)$ is q$^-$-rational such that $\overline{\pi}(S_2)$ is finite (i.e., S_2 is obtained without usage of the *-operator) then $S_1 \cdot \mathcal{Q}(A,U) \cdot S_2$ is q-rational

▶ **Example 4.8.** Let $S = \{q \in \mathcal{Q}(A,U) \mid \pi(q) \in (ab)^*, \overline{\pi}(q) = b\}$. Then S is q-rational since we have $S = \overline{\pi}^{-1}(b) \cap (\pi^{-1}(a) \cdot \pi^{-1}(b))^*$. Note that the class of q-rational subsets also is closed under intersection due to Rule 2, i.e., this class is a Boolean algebra.

At first sight, the choice of Rule 3 seems to be some kind of random. But we can remove neither the factor $\mathcal{Q}(A,U)$, which appears as separator in this product, nor the finiteness of $\overline{\pi}(S_2)$. Additionally, we cannot simply remove this rule since the set $\{[a\overline{a}]\}$ cannot be built by application of the Rules 1 and 2, only.

Now we can prove the implication "A⇒B" in Theorem 4.1. To do this, we utilize Theorem 4.63. Concretely, we do this by induction on the syntax tree of such kind of expression that each recognizable subset is q-rational. The most complicated case in this proof is to show that Ω_ℓ is q-rational. For this proof we need the following lemma:

▶ **Lemma 4.9.** *Let $\ell \in \mathbb{N}$, $q \in \mathcal{Q}(A,U)$ and $u = a_1 \dots a_\ell \in A^*$. Then we have $u \leq_U \pi(q)$ and $\overline{\pi}_2(q) \in A^*u$ if, and only if, $q \in \pi^{-1}\Big(\prod_{i=1}^{\ell}(A \setminus U)^* a_i\Big) \cdot \mathcal{Q}(A,U) \cdot \overline{\pi}^{-1}(u)$.* ◀

Finally, we can state the following implication:

▶ **Proposition 4.10.** *Let $S \subseteq \mathcal{Q}(A,U)$ be recognizable. Then S is q-rational.*

Proof. We use Theorem 4.63 to prove the claim by induction. At first, if $S = \pi^{-1}(R)$ or $S = \overline{\pi}^{-1}(R)$ where $R \subseteq A^*$ is regular, then S is q-rational by Observation 4.7.

Next, let $\ell \in \mathbb{N}$ and $S = \Omega_\ell$. Then by Observation 4.5 and Lemma 4.9 we have

$$\Omega_\ell = \bigcap_{u \in A^{\leq \ell}} \left(\mathcal{Q}(A,U) \setminus \left(\pi^{-1}(W_u A^*) \cap \overline{\pi}^{-1}(A^*u)\right) \cup \pi^{-1}(W_u) \cdot \mathcal{Q}(A,U) \cdot \overline{\pi}^{-1}(u)\right),$$

where $W_u = \prod_{i=1}^{k}(A \setminus U)^* a_i$ with $u = a_1 \dots a_k$. Since the sets $\pi^{-1}(W_u A^*)$, $\overline{\pi}^{-1}(A^*u)$, $\pi^{-1}(W_u)$, and $\overline{\pi}^{-1}(u)$ are q-rational by Observation 4.7, Ω_ℓ is q-rational as well.

Finally, the class of q-rational subsets is closed under Boolean operations. ◀

4.3 From Q-Rational Subsets to Logic

The second implication from Theorem 4.1 states that each q-rational subset is definable in a special monadic second-order logic which we call $\mathsf{MSO_q}$. Here, we try to exhibit the knowledge from the preceding subsection such that this logic defines exactly the recognizable subsets. In fact, we have to add some modifications to Büchi's MSO-logic from [4]. At first, we should understand $p \leq^w q$ as follows: the letter a on position p in w cannot be moved to the right of the letter b on position q without violating any of the rules from Definition 2.1 (recall that $\mathcal{R}$ only swaps letters). In other words, for any $v \in [w]$ the letter a appears left from b in v. Additionally, we have to restrict comparisons of write and read operations:

▶ Remark. It is not possible to compare arbitrary letters in w without any restrictions. For example, let

$$\phi = \exists x, y\colon (Q_A(x) \wedge \forall z\colon (Q_A(z) \to z \leq x) \wedge Q_{\overline{A}}(y) \wedge \forall z\colon (Q_{\overline{A}}(z) \to y \leq z) \wedge \neg x \leq y),$$

i.e., $\underline{w}$ satisfies ϕ iff the first read action can be moved to the right of the last write action. Then we have $L(\phi) \cap \overline{a}^* a^* \overline{a}^* = \{\overline{a}^k a^\ell \overline{a}^m \mid k = 0 \text{ or } m \geq \ell\}$. Since this language is not regular, the subset of $\mathcal{Q}(A,U)$ of the elements satisfying ϕ is not recognizable either.

By $\mathsf{FO_q}$ we denote the set of all first-order formulas build up from the atomic formulas of the form $x = y$, $x <_+ y$, $x <_- y$, $P_\ell(x)$ for $\ell \in \mathbb{N}_+$, and $Q_a(x)$ for $a \in A$ where x and y are variables. Additionally, by $\mathsf{MSO_q}$ we denote the monadic second-order extension of $\mathsf{FO_q}$.

Now let $w = a_1 \dots a_n \in \Sigma^*$. The *plq model* for w is the relational structure $\widetilde{w} := (\mathsf{dom}(w), <_+^w, <_-^w, (P_\ell^w)_{\ell \in \mathbb{N}_+}, (Q_a^w)_{a \in \Sigma})$ where $\mathsf{dom}(w) = \{1, \dots, n\}$, $Q_a^w = \{i \mid a_i = a\}$, $<_+^w$ and $<_-^w$ are the natural orderings on $Q_A^w = \bigcup_{a \in A} Q_a^w$ and $Q_{\overline{A}}^w$, respectively, and

$$P_\ell^w = \{i \in Q_A^w \mid \forall v_1, v_2 \in \Sigma^*\colon (w \equiv v_1 v_2 \wedge \pi(v_1) = \pi(a_1 \dots a_i)) \to |\overline{\pi}(v_2)| < \ell\},$$

i.e., we have $i \in P_\ell^w$ iff $a_i \in A$ and the ℓth last read action in w is left from a_i and cannot be moved to the right of a_i. This is conform to the approaches known from [4, 7] since the relations $<_+^w$, $<_-^w$, and P_ℓ^w specify which letter have to appear to the left of another one in any word equivalent to w. Hence, we can infer that $\widetilde{w}$ identifies the equivalence class $[w]$:

▶ **Lemma 4.11.** *Let $v, w \in \Sigma^*$. Then we have $v \equiv w$ if, and only if, $\widetilde{v} \cong \widetilde{w}$.* ◀

Therefore, we can define the *plq model* $\widetilde{q} := \widetilde{\mathsf{nf}(q)}$ for $q \in \mathcal{Q}(A, U)$.

Now let $\phi \in \mathsf{MSO_q}$. The set defined by ϕ is $S(\phi) = \{q \in \mathcal{Q}(A,U) \mid \widetilde{q} \models \phi\}$. We say that $S \subseteq \mathcal{Q}(A,U)$ is $\mathsf{MSO_q}$-*definable* ($\mathsf{FO_q}$-*definable*) if there is $\phi \in \mathsf{MSO_q}$ ($\phi \in \mathsf{FO_q}$, respectively) with $S = S(\phi)$.

▶ Remark. The sets P_ℓ^w also are conform to the special product in the definition of q-rational subsets into logics. In particular, we have $S(\exists x\colon \neg P_\ell(x)) = \pi^{-1}(A^+) \cdot \mathcal{Q}(A,U) \cdot \overline{\pi}^{-1}(A^\ell)$.

Now we prove that each q-rational subset is $\mathsf{MSO_q}$-definable. In the proof of implication "B⇒C" in Theorem 4.1 we need the following notion: Let $\phi, \xi(x) \in \mathsf{MSO}$. Then there is a formula $\phi|_\xi \in \mathsf{MSO}$ which restricts the quantifiers in ϕ to values satisfying $\xi(x)$. Thereby, we have $\phi|_\xi \in \mathsf{FO}$ iff $\phi, \xi \in \mathsf{FO}$.

Finally, we can state:

▶ **Proposition 4.12.** *Let $S \subseteq \mathcal{Q}(A,U)$ be q-rational. Then S is $\mathsf{MSO_q}$-definable.*

Proof. If S is q^+-rational then we have $S = \pi^{-1}(R)$ for some regular $R \subseteq A^*$. By [4] there is an MSO-formula ϕ with $L(\phi) = R$. Then by replacing of all occurrences of $<$ in ϕ by $<_+$ we obtain an $\mathsf{MSO_q}$-formula ϕ' with $S(\phi'|_{Q_A(x)}) = \pi^{-1}(L(\phi)) = S$.

Similarly, we can prove that S is $\mathsf{MSO_q}$-definable if S is q^--rational (here, we replace $<$ by $<_-$ and restrict to $Q_{\overline{A}}$).

If $S = S_1 \cup S_2$ or $S = \mathcal{Q}(A,U) \setminus S_1$, where S_1, S_2 are q-rational there are $\phi_1, \phi_2 \in \mathsf{MSO_q}$ with $S(\phi_1) = S_1$ and $S(\phi_2) = S_2$. Then we have $S = S(\phi_1 \vee \phi_2)$ and $S = S(\neg\phi_1)$, respectively.

Finally, let $S = \pi^{-1}(R) \cdot \mathcal{Q}(A,U) \cdot \overline{\pi}^{-1}(F)$ where $R \subseteq A^*$ is regular and $F \subseteq A^*$ is finite. W.l.o.g. we can assume that $F = \{w\}$ holds. Then there are $\mathsf{MSO_q}$-formulas ϕ_R and ϕ_F defining $\pi^{-1}(R)$ and $\overline{\pi}^{-1}(F)$, respectively. Set

$$\phi := \exists x_1, x_2\colon \phi_R|_{x \leq_+ x_1} \wedge \phi_F|_{x_2 \leq_- x} \wedge \neg P_{|w|}(x_1)\,.$$

Then we have $S = S(\phi)$. ◀

4.4 From Logic to Recognizability

Finally, we have to prove that each $\mathsf{MSO_q}$-definable subset is recognizable. To do this, we utilize Theorem 4.62. In other words, given $\phi \in \mathsf{MSO_q}$ we construct a formula $\psi \in \mathsf{MSO}$ such that $\eta^{-1}(S(\phi)) \cap \overline{A}^* A^* \overline{A}^* = L(\psi) \cap \overline{A}^* A^* \overline{A}^*$ holds. Since the right-hand side of this equation is regular by [4], we can infer that $S(\phi)$ is recognizable.

The translation of formula $P_\ell(x)$ is the most complicated case in our construction since write and read actions commute in certain contexts given in Definition 2.1. Concretely, we will translate $\neg P_\ell(x)$ since it seems to be easier to understand. Hence, we start with this case. At first, we prove that there is an FO-formula describing the words in which the last ℓ read actions are U-prefixes of the write actions:

▶ **Lemma 4.13.** *Let $\ell \in \mathbb{N}$. There is a sentence $\mathsf{overlap}_\ell \in \mathsf{FO}$ such that $w \in L(\mathsf{overlap}_\ell)$ if, and only if, there is $u \in A^\ell$ with $u \leq_U \pi(w)$ and $\overline{\pi}_2(w) \in A^* u$ for any $w \in \Sigma^*$.*

Proof. There is an NFA $\mathcal{A}$ which guesses the last ℓ read actions, verifies afterwards whether these are a U-prefix of the write actions, and checks whether each of the last ℓ read actions appear after their corresponding write action. This NFA can be constructed without usage of any counters. Hence, its accepted language is aperiodic. By [19] there is a formula $\mathsf{overlap}_\ell \in \mathsf{FO}$ with $L(\mathsf{overlap}_\ell) = L(\mathcal{A})$. ◀

Now let $w \in \overline{A}^* A^* \overline{A}^*$ and $p \in \mathsf{dom}(w)$. Then we express $p \notin P_\ell^w$ as follows:

If we have $p \in Q_{\overline{A}}^w$ we are ready. So, assume $p \in Q_A^w$ from now on. At first, we choose the last ℓ read actions from w. Let $q_1, \ldots, q_\ell$ be their positions.

If $p < q_1$ then we are ready. So, assume $q_1 < p$ from now on. Then there are two words $u, v \in \Sigma^*$ such that $w \equiv uv$, u ends with the letter on position p in w, and v starts with the letter on position q_1 in w. Since $\equiv$ is a congruence we can assume that $v = \overline{\overline{\pi}_1(v)}\pi(v)\overline{\overline{\pi}_2(v)}$ holds. Let $\overline{\pi}(v) = b_1 \ldots b_\ell$. Then $\overline{b_i}$ can be moved to the left-hand side of the letter on position p in w if, and only if, uv does not satisfy $\mathsf{overlap}_{\ell-i+1}$.

Finally, there may be some letters $\overline{b_i}$ from $\overline{\overline{\pi}_1(v)}$ that can be moved to the right in w. This is possible if, and only if, one of the following two cases hold: on the one hand, this is possible if $b_i \ldots b_\ell \not\leq_U \pi(w)$. On the other hand, if $b_i \ldots b_\ell \leq_U \pi(w)$ and the write action corresponding to $\overline{b_i}$ appears right from position p in w.

All of the above mentioned requirements can be expressed in MSO-formulas. Hence, we can construct $\mathsf{co\text{-}P}_\ell(x) \in \mathsf{MSO}$ such that $\underline{w} \models \mathsf{co\text{-}P}_\ell[p]$ if, and only if, $p \notin P_\ell^w$. Therefore, we can state the following:

▶ **Proposition 4.14.** *Let $S \subseteq \mathcal{Q}(A, U)$ be $\mathsf{MSO_q}$-definable. Then S is recognizable.*

Proof. Let $S \subseteq \mathcal{Q}(A, U)$ be $\mathsf{MSO_q}$-definable. Then there is $\phi \in \mathsf{MSO_q}$ with $S = S(\phi)$. We construct $\phi' \in \mathsf{MSO}$ by the following modifications of ϕ:

- replace "$x <_+ y$" by "$x < y \wedge Q_A(x) \wedge Q_A(y)$"
- replace "$x <_- y$" by "$x < y \wedge Q_{\overline{A}}(x) \wedge Q_{\overline{A}}(y)$"
- replace "$P_\ell(x)$" by "$\neg\mathsf{co\text{-}P}_\ell[x]$"

Then we can prove that $\widetilde{w} \models \phi$ if, and only if, $\underline{w} \models \phi'$ for any $w \in \overline{A}^* A^* \overline{A}^*$. Hence, by Büchi's Theorem [4] $\eta^{-1}(S) \cap \overline{A}^* A^* \overline{A}^*$ is regular, i.e., S is recognizable due to Theorem 4.6. ◀

5 Characterizations of the Aperiodic Subsets

In the previous section we have seen a Kleene- and Büchi-type characterization of the recognizable subsets in the plq monoid. Another more involved task is to describe the aperiodic subsets in the plq monoid. Schützenberger has proven in [25] that the aperiodic subsets in the free monoid are exactly the star-free languages. This result gives us a decision procedure to decide whether a given regular language is star-free. Another similar result for trace monoids can be found in [10]. These two results cannot be translated to plq monoids since the class of aperiodic subsets is not closed under product. Though, we will see that we can restrict the use of the product to describe exactly the aperiodic subsets of the plq monoid.

Another characterization of the aperiodic languages was proven by [19]: similar to Büchi's Theorem [4] McNaughton and Papert proved that these are exactly the FO-definable languages. Here, we will see that analogously the aperiodic subsets in the plq monoid are the $\mathsf{FO_q}$-definable subsets.

Before we give these characterizations we have to define the restriction of star-freeness. We say that a subset of $\mathcal{Q}(A, U)$ is *q-star-free* if it can be constructed by the Rules 1-3 in which we replace "$S = S_1^*$" by "$S = \mathcal{Q}(A, U) \setminus S_1$" in the rules 2 and (2^-).

Similarly, to Theorem 4.1 we can state and prove the following result:

▶ **Theorem 5.1.** *Let A be an at least binary alphabet, $U \subseteq A$, and $S \subseteq \mathcal{Q}(A, U)$. Then the following are equivalent:*
(A) *S is aperiodic.*
(B) *L is q-star-free.*
(C) *L is FO_q-definable.* ◀

References

1 Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Inf. Comput.*, 127(2):91–101, 1996. doi:10.1006/inco.1996.0053.
2 Jean Berstel. *Transductions and Context-Free Languages*. Teubner Studienbücher, 1979. doi:10.1007/978-3-663-09367-1.
3 Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983. doi:10.1145/322374.322380.
4 J. Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960. doi:10.1002/malq.19600060105.
5 Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Inf. Comput.*, 124(1):20–31, 1996. doi:10.1006/inco.1996.0003.
6 Marek Chrobak and Wojciech Rytter. Unique deciperability for partially commutative alphabet (extended abstract). In Jozef Gruska, Branislav Rovan, and Juraj Wiedermann, editors, *Mathematical Foundations of Computer Science 1986, Bratislava, Czechoslovakia, August 25-29, 1996, Proceedings*, volume 233 of *Lecture Notes in Computer Science*, pages 256–263. Springer, 1986. doi:10.1007/BFb0016249.
7 Volker Diekert and Grzegorz Rozenberg. *The book of traces*. World scientific, 1995. doi:10.1142/9789814261456.
8 Alain Finkel. Decidability of the termination problem for completely specified protocols. *Distributed Computing*, 7(3):129–135, 1994. doi:10.1007/BF02277857.
9 Alan Gibbons and Wojciech Rytter. On the decidability of some problems about rational subsets of free partially commutative monoids. *Theor. Comput. Sci.*, 48(3):329–337, 1986. doi:10.1016/0304-3975(86)90101-5.
10 Giovanna Guaiana, Antonio Restivo, and Sergio Salemi. On aperiodic trace languages. In Christian Choffrut and Matthias Jantzen, editors, *STACS 91, 8th Annual Symposium on Theoretical Aspects of Computer Science, Hamburg, Germany, February 14-16, 1991, Proceedings*, volume 480 of *Lecture Notes in Computer Science*, pages 76–88. Springer, 1991. doi:10.1007/BFb0020789.
11 Martin Huschenbett, Dietrich Kuske, and Georg Zetzsche. The monoid of queue actions. *Semigroup forum*, 95(3):475–508, 2017. doi:10.1007/s00233-016-9835-4.
12 Mark Kambites. Formal languages and groups as memory. *Communications in Algebra*, 37(1):193–208, 2009. doi:10.1080/00927870802243580.
13 Stephen C. Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, 1956.
14 Chris Köcher. Einbettungen in das Transformationsmonoid einer vergesslichen Warteschlange. Master's thesis, TU Ilmenau, 2016. doi:10.13140/RG.2.2.21984.99842.
15 Chris Köcher, Dietrich Kuske, and Olena Prianychnykova. The transformation monoid of a partially lossy queue. *RAIRO - Theoretical Informatics and Applications*, 2018. To appear.
16 Markus Lohrey. The rational subset membership problem for groups: a survey. In *Groups St Andrews*, volume 422, pages 368–389. Cambridge University Press, 2013. doi:10.1017/CBO9781316227343.024.

17 Benoît Masson and Philippe Schnoebelen. On verifying fair lossy channel systems. In Krzysztof Diks and Wojciech Rytter, editors, *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, volume 2420 of *Lecture Notes in Computer Science*, pages 543–555. Springer, 2002. doi:10.1007/3-540-45687-2_45.

18 J. D. McKnight. Kleene quotient theorems. *Pacific Journal of Mathematics*, 14(4):1343–1352, 1964.

19 Robert McNaughton and Seymour A. Papert. *Counter-Free Automata*, volume 65. The MIT Press, 1971.

20 Anca Muscholl and Holger Petersen. A note on the commutative closure of star-free languages. *Inf. Process. Lett.*, 57(2):71–74, 1996. doi:10.1016/0020-0190(95)00187-5.

21 Edward Ochmański. Regular behaviour of concurrent systems. *Bulletin of the EATCS*, 27:56–67, 1985.

22 Jean-Éric Pin. Mathematical foundations of automata theory. *Lecture notes LIAFA, Université Paris*, 7, 2010.

23 Emil L. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946.

24 Elaine Render and Mark Kambites. Rational subsets of polycyclic monoids and valence automata. *Inf. Comput.*, 207(11):1329–1339, 2009. doi:10.1016/j.ic.2009.02.012.

25 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965. doi:10.1016/S0019-9958(65)90108-7.

26 Wolfgang Thomas. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages: Volume 3 Beyond Words*, pages 389–455. Springer, 1997. doi:10.1007/978-3-642-59126-6_7.

Relations Between Greedy and Bit-Optimal LZ77 Encodings

Dmitry Kosolobov
University of Helsinki, Helsinki, Finland
dkosolobov@mail.ru

Abstract

This paper investigates the size in bits of the LZ77 encoding, which is the most popular and efficient variant of the Lempel–Ziv encodings used in data compression. We prove that, for a wide natural class of variable-length encoders for LZ77 phrases, the size of the greedily constructed LZ77 encoding on constant alphabets is within a factor $O(\frac{\log n}{\log\log\log n})$ of the optimal LZ77 encoding, where n is the length of the processed string. We describe a series of examples showing that, surprisingly, this bound is tight, thus improving both the previously known upper and lower bounds. Further, we obtain a more detailed bound $O(\min\{z, \frac{\log n}{\log\log z}\})$, which uses the number z of phrases in the greedy LZ77 encoding as a parameter, and construct a series of examples showing that this bound is tight even for binary alphabet. We then investigate the problem on non-constant alphabets: we show that the known $O(\log n)$ bound is tight even for alphabets of logarithmic size, and provide tight bounds for some other important cases.

2012 ACM Subject Classification Mathematics of computing → Coding theory

Keywords and phrases Lempel–Ziv, LZ77 Encoding, Greedy LZ77, Bit Optimal LZ77

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.46

1 Introduction

The Lempel–Ziv encoding [23] (LZ77 for short) is one of the most popular and efficient compression techniques used in data compression, stringology, and algorithms in general. The LZ77 encoding lies at the heart of common compressors such as `gzip`, `7zip`, `pkzip`, `rar`, etc. and serves as a basis for modern compressed text indexes on highly repetitive data (e.g., see [12, 15, 18]).

Numerous papers on LZ77 have been published during the last 40 years. In these works, it was proved that LZ77 is superior compared to many other compression schemes both in practice and in theory. For instance, in [14, 24, 22] it was shown that LZ77 is asymptotically optimal with respect to different entropy-related measures; further, in [4] it was proved that many other reference based encoders (including LZ78 [24]) use polynomially (in the length of the uncompressed data) more space than LZ77 in the worst case and, in a sense, are never significantly better than LZ77. However, many problems related to LZ77 are still not completely solved. In this paper we investigate how good is the popular greedy LZ77 encoder in a class of practically motivated models with variable-length encoders for LZ77 phrases; to formulate the problem that we study more accurately, let us first discuss what is known about different LZ77 encoders.

LZ77 is a dictionary based compression scheme that replaces a string with phrases that are actually references to strings in a dictionary. Each phrase of an LZ77 encoding can be viewed as a triple $\langle d, \ell, c\rangle$, where ℓ is the length of the phrase, d is the distance to a string of length $\ell-1$ from the dictionary such that this string is a prefix of the phrase, and c is the last letter of the phrase (the precise definition follows); we use the definition from [23] but all our results can be adapted for the version of LZ77 from [21], in which phrases are

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 46; pp. 46:1–46:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


encoded by pairs $\langle d, \ell \rangle$ (throughout the paper, we provide the reader with separate remarks in cases where such adaptation is not straightforward). The same string can have many different LZ77 encodings. It is well known that the greedily constructed LZ77 encoding, which builds the encoding from left to right making each phrase as long as possible during this process, is optimal in the sense that it produces the minimal number of phrases among all LZ77 encodings of this string (see [4, 20, 21]). The same optimality property holds for the versions of LZ77 with "sliding window" [9], which is a restriction that is important for practical applications.

However, in practice, compressors usually use variable-length encoders for phrases and, in this case, it is not clear whether the greedy LZ77 encoder is optimal in the sense that it outputs the minimal number of bits. The question of finding an optimal LZ77 encoding for variable-length phrase encoders was raised in [19] and the first attempts to solve this problem were given in [11]. The authors of [11] also conducted the first theoretical studies to find how bad is the greedy LZ77 encoding compared to an optimal LZ77 encoding. Such questions make sense only if we state formally which kinds of phrase encoders are used in the LZ77 encoder. As in [11], we investigate encoders that encode each phrase $\langle d, \ell, c \rangle$ using $\Theta(\log d + \log \ell + \log c)$ bits[1] (see a more formal discussion below). This class of phrase encoders includes a broad range of practically used encoders and, among others, Elias's [10] and Levenshtein's [17] encoders, which produce asymptotically optimal universal codes for the numbers d, ℓ, c; we refer the reader to [11] for further discussions on the motivation.

In the described model, there are two ways how to optimize the size of the produced LZ77 encoding. The first way is to minimize d in the triples $\langle d, \ell, c \rangle$. This problem was addressed already in [11] for the greedy LZ77 encoder, where one must find the rightmost occurrence of the referenced part of each phrase; several improvements on this result of [11] and related questions were given in [1, 2, 3, 8, 16]. The second way is to consider both parameters ℓ and d, i.e., to build an optimal LZ77 encoding. There are very few works in this direction (see [7] and [11]) and there is still a room for improvements in such results. Due to the overall difficulty of the problem of finding an optimal LZ77 encoding, real compressors usually construct an LZ77 encoding greedily. Thus, this raises the following question: how bad can the produced greedy LZ77 encoding be compared to an optimal LZ77 encoding?

For a given string of length n, denote by $\mathsf{LZ_{gr}}$ and $\mathsf{LZ_{opt}}$ the sizes in bits of, respectively, the greedily constructed and an optimal LZ77 encodings from the special class of encodings that we consider in this paper (see clarifications in Section 2). We investigate the ratio $\frac{\mathsf{LZ_{gr}}}{\mathsf{LZ_{opt}}}$. Upper bounds on this ratio are provided in terms of the parameters n, z, and σ, where z is the number of phrases in the greedy LZ77 encoding of the considered string (it is well known that any other LZ77 encoding contains at least z phrases; see [4, 20, 21]) and σ is the alphabet size. We are also interested in upper bounds that use only the parameter n. In [11] it was proved that $\frac{\mathsf{LZ_{gr}}}{\mathsf{LZ_{opt}}} = O(\log n)$ and there is a series of examples on which $\frac{\mathsf{LZ_{gr}}}{\mathsf{LZ_{opt}}} = \Omega(\frac{\log n}{\log \log n})$. In this paper we improve these results and our bounds in many cases are tight in the sense that there are series of examples on which these bound are attained; our main contributions are summarized in Table 1.

First, we study the case of constant alphabets and completely solve it. Namely, in Theorem 7, we find the following detailed upper bound on the ratio $\frac{\mathsf{LZ_{gr}}}{\mathsf{LZ_{opt}}}$ (note that this bound is also applicable for arbitrary alphabets): $\frac{\mathsf{LZ_{gr}}}{\mathsf{LZ_{opt}}} = O(\min\{z, \frac{\log n}{\log \log_\sigma z}\})$. In the case of constant alphabets this upper bound degenerates to $O(\min\{z, \frac{\log n}{\log \log z}\})$. In Theorem 10 we

[1] Throughout the paper all logarithms have base 2 if it is not explicitly stated otherwise.

Table 1 Upper bounds on $\mathsf{LZ}_{\mathsf{gr}}/\mathsf{LZ}_{\mathsf{opt}}$; tight bounds are denoted by Θ.

	parameter n	parameters n, z, σ
$\sigma = O(1)$	$\Theta(\frac{\log n}{\log\log\log n})$	$\Theta(\min\{z, \frac{\log n}{\log\log z}\})$
arbitrary σ	$\Theta(\log n)$	$O(\min\{z, \frac{\log n}{\log\log_\sigma z}\})$

construct a series of examples on the binary alphabet showing that this simplified bound is tight, thus closing the problem for constant alphabets. Theorem 10 actually provides a more elaborate lower bound $\Omega(\min\{z, \frac{\log n}{\log\log_\sigma z + \log\sigma}\})$, which is applicable for arbitrary alphabets. From these general results, we deduce in Corollary 8 that $\frac{\mathsf{LZ}_{\mathsf{gr}}}{\mathsf{LZ}_{\mathsf{opt}}} = O(\frac{\log n}{\log\log\log n})$ for constant alphabets, and this upper bound is tight.

Then, we consider the case of arbitrary alphabets. It is shown in Theorem 12 that the upper bound $O(\log n)$ on the ratio $\frac{\mathsf{LZ}_{\mathsf{gr}}}{\mathsf{LZ}_{\mathsf{opt}}}$ is tight even if the input alphabet has logarithmic size. Thus, we solve the problem in the general case and find that the tight upper bounds, expressed in terms of n, for constant and arbitrary alphabets differ by $\Theta(\log\log\log n)$ factor.

As a side note, for polylogarithmic alphabets and $z \geq 2^{\log^\epsilon n}$, where $\epsilon > 0$ is an arbitrary constant, we obtain in Corollary 11 the upper bound $O(\frac{\log n}{\log\log n})$ and show that this bound is tight for such alphabets and such z. Informally, the strings for which the condition $z \geq 2^{\log^\epsilon n}$ holds (which includes the case $z \geq n^\delta$, where $\delta > 0$ is an arbitrary constant) can be called "non-extremely compressible" strings. Thus, we, in a sense, solve the problem in the arguably most important case of "non-extremely compressible" strings drawn from polylogarithmic alphabets.

The paper is organized as follows. In the following Section 2 we introduce some basic notions used throughout the text and, in particular, formally define LZ77 parsings and encodings. Section 3 describes a detailed upper bound on the ratio of the sizes in bits of the greedy and optimal LZ77 encodings. In Section 4 it is shown that, on constant alphabets, this bound is tight. The material of these two sections provides a complete solution of the problem for constant alphabets, which turns out to be quite simple. We then consider arbitrary alphabets in Section 5 and find tight bounds for several important cases, including the general case of arbitrary alphabet and arbitrary z, for which, as it turns out, the known $O(\log n)$ bound is tight. Finally, we conclude with some remarks and open problems in Section 6.

2 Preliminaries

A *string* s over an alphabet Σ is a map $\{1, 2, \ldots, n\} \to \Sigma$, where n is referred to as the *length of* s, denoted by $|s|$. In this paper we assume that the alphabet is a set of non-negative integers that are less than or equal to n, which is a common and natural assumption in the problem under investigation. We write $s[i]$ for the ith letter of s and $s[i..j]$ for $s[i]s[i+1]\cdots s[j]$. A string u is a *substring* of s if $u = s[i..j]$ for some i and j; the pair (i, j) is not necessarily unique and we say that i specifies an *occurrence* of u in s starting at position i. A substring $s[1..j]$ (resp., $s[i..n]$) is a *prefix* (resp. *suffix*) of s. We say that substrings $s[i..j]$ and $s[i'..j']$ *overlap* if $j \geq i'$ and $i \leq j'$. For any i, j, the set $\{k \in \mathbb{Z} \colon i \leq k \leq j\}$ (possibly empty) is denoted by $[i..j]$.

An *LZ77 parsing* of a given string s is a parsing $s = f_1 f_2 \cdots f_z$ such that all the strings $f_1, \ldots, f_z$ (called *phrases*) are non-empty and, for any $i \in [1..z]$, either f_i is a letter, or $|f_i| > 1$ and the string $f_i[1..|f_i|-1]$ has an earlier occurrence starting at some position $j \leq |f_1 f_2 \cdots f_{i-1}|$ (note that this occurrence can overlap f_i).

The *greedy LZ77 parsing* is a special LZ77 parsing built by the greedy procedure that constructs all phrases from left to right by choosing each phrase f_i as the longest substring starting at given position such that $f_i[1..|f_i|-1]$ has an earlier occurrence in the string (see [23]). For instance, the greedy LZ77 parsing of the string $s = ababbbbaba$ is $a.b.ababb.baba$. The following lemma is straightforward.

▶ **Lemma 1.** *All phrases in the greedy LZ77 parsing of a given string (except, possibly, for the last phrase) are distinct.*

It is also well-known that, for a given string, the greedy LZ77 parsing has the minimal number of phrases among all LZ77 parsings (e.g., see [4, 20, 21]). This implies that, when each phrase of the parsing is encoded by a fixed number of bits, the greedy LZ77 parsing is optimal, i.e., it produces an encoding of the minimal size in bit. However, the greedy LZ77 parsing does not necessarily produce an encoding of the minimal size when one uses a variable-length encoder for phrases; the latter is usually the case in most common compressors. Let us clarify what kinds of variable-length phrase encoders we are to consider in this paper.

A given LZ77 parsing $f_1 f_2 \cdots f_z$ is encoded as follows. Each phrase f_i is represented by a triple $\langle d, \ell, c\rangle$, where $\ell = |f_i|$, $c = f_i[|f_i|]$, and $d = |f_1 f_2 \cdots f_{i-1}| - j$ for j that is the position of an earlier occurrence of $f_i[1..|f_i|-1]$ (assuming that $d = 0$ if $|f_i| = 1$). We choose three encoders e_d, e_ℓ, e_c, each of which maps non-negative integers to bit strings. We then transform each triple $\langle d, \ell, c\rangle$ into the binary string $e_d(d)e_\ell(\ell)e_c(c)$ and concatenate all these binary strings, thus producing an *LZ77 encoding* corresponding to the given LZ77 parsing.

In this paper we consider only encoders e_d, e_ℓ, e_c that map any positive integer x to a bit string of length $\Theta(\log(x+1))$. This family of encoders includes most widely used encoders such as Elias's [10] and Levenshtein's [17] ones (see [11] for further motivation). We fix three encoders e_d, e_ℓ, e_c satisfying the above property and, hereafter, assume that all considered LZ77 encodings are obtained using these e_d, e_ℓ, e_c.

We say that an LZ77 encoding is *optimal* if it has the minimal size in bits. It is shown below that, unlike the case of fixed-length phrase encoders, for the family of phrase encoders under investigation, the LZ77 encoding generated by the greedy LZ77 parsing (which is called the *greedy LZ77 encoding*) is not necessarily optimal. Among all possible greedy LZ77 encodings we always consider those that occupy the minimal number of bits; usually, such encoding is obtained by the minimization of the numbers d in the triples $\langle d, \ell, c\rangle$ representing the phrases of the greedy LZ77 parsing.

▶ Remark. Most common compressors actually use a different variant of the LZ77 parsing (which was introduced in [21]), defining each phrase f_i as either a letter or a string that has an earlier occurrence (note that in the definition of LZ77 parsings only the prefix $f_i[1..|f_i|-1]$ of f_i must have an earlier occurrence). We call this variant a *nonclassical LZ77 parsing* (as it differs from the original parsing proposed in [23]). The *greedy nonclassical LZ77 parsing* is defined by analogy with the greedy LZ77 parsing. In encoding corresponding to a nonclassical LZ77 parsing each phrase is represented either by a pair $\langle d, \ell\rangle$ that is defined analogously to the triples $\langle d, \ell, c\rangle$, or by one letter. This variant of LZ77 is very similar to the one that we investigate and, moreover, all our results can be adapted for this variant. In the sequel, we provide separate remarks that explicitly show how to generalize our results to nonclassical LZ77 parsings if it is not straightforward.

3 Upper Bound

Our proof of the upper bound on the ratio between the sizes of the greedy and optimal LZ77 encodings is as follows: first, we obtain an upper bound U on the size of the greedy LZ77

encoding, then we find a lower bound L on the size of any LZ77 encoding, and finally, we derive the estimation $\frac{U}{L}$ on the ratio. The details follow.

Let s be a string of length n. Recall that any letter of s is an integer from the range $[0..n]$. Based on the above mentioned properties of the phrase encoders e_d, e_ℓ, e_c, one can easily show that each phrase of any LZ77 encoding of s occupies $O(\log n)$ bits. Therefore, we obtain the following upper bound on the size of the greedy LZ77 encoding.

▶ **Lemma 2.** *Let $\mathsf{LZ}_{\mathsf{gr}}$ be the size in bits of the greedy LZ77 encoding of a given string of length n. Then, we have $\mathsf{LZ}_{\mathsf{gr}} = O(z \log n)$, where z is the number of phrases in the encoding.*

The lower bound on any LZ77 encoding is more complicated. Lemmas 3, 4, 5 below are well known but we, nevertheless, provide their proofs for the sake of completeness.

▶ **Lemma 3.** *For any positive integers $t, t_1, \ldots, t_k$ such that $\sum_{i=1}^{k} t_i \geq t$, we have $\sum_{i=1}^{k} \log t_i \geq \log(t-k+1)$.*

Proof. Note that $\sum_{i=1}^{k} \log t_i = \log \prod_{i=1}^{k} t_i$. Since for any t_j and $t_{j'}$ such that $t_j \geq t_{j'}$, we have $(t_j+1)(t_{j'}-1) = t_j t_{j'} - (t_j - t_{j'} + 1) < t_j t_{j'}$, the product $\prod_{i=1}^{k} t_i$ is minimized when $t_1 = t-k+1$ and $t_2 = t_3 = \cdots = t_k = 1$ (recall that every number t_i must be a positive integer). Therefore, we obtain $\sum_{i=1}^{k} \log t_i \geq \log(t-k+1)$. ◀

▶ **Lemma 4.** *Any phrase of an LZ77 parsing of a string can overlap with at most two phrases of the greedy LZ77 parsing of the same string.*

Proof. Suppose, for the sake of contradiction, that a phrase f of an LZ77 parsing overlaps with at least three phrases of the greedy LZ77 parsing. Then, $f[1..|f|-1]$ must contain a phrase f' of the greedy LZ77 parsing as a proper substring. But then the string f' occurs in an earlier occurrence of the string $f[1..|f|-1]$ and, therefore, the greedy construction procedure could choose a longer phrase during the construction of the phrase f', which is a contradiction. ◀

▶ **Lemma 5.** *In the greedy LZ77 parsing of any string of length n over an alphabet of size $\sigma \geq 2$, at least $z - 2\sqrt{z}$ phrases have length $\geq \frac{1}{2}\log_\sigma z$, where z is the number of phrases.*

Proof. Denote by $f_1 f_2 \cdots f_z$ the greedy LZ77 parsing of a given string of length n over an alphabet of size σ. By Lemma 1, all the phrases $f_1, \ldots, f_{z-1}$ are distinct. Therefore, for any $\ell > 0$, at most $\sum_{i=0}^{\ell} \sigma^i = \frac{\sigma^{\ell+1}-1}{\sigma-1}$ of these phrases have length at most ℓ. Since for any $\ell < \frac{1}{2}\log_\sigma z$, we have $\sum_{i=0}^{\ell} \sigma^i < \frac{\sqrt{z}\sigma-1}{\sigma-1}$, the number of phrases with length at least $\frac{1}{2}\log_\sigma z$ must be greater than $(z-1) - \frac{\sqrt{z}\sigma-1}{\sigma-1}$. Thus, it remains to prove that $1 + \frac{\sqrt{z}\sigma-1}{\sigma-1} \leq 2\sqrt{z}$. It is easy to show that, for $\sigma \geq 2$, the function $\frac{\sqrt{z}\sigma-1}{\sigma-1}$ decreases as σ grows. Hence, we deduce $1 + \frac{\sqrt{z}\sigma-1}{\sigma-1} \leq 1 + \frac{2\sqrt{z}-1}{2-1} = 2\sqrt{z}$. ◀

▶ **Lemma 6.** *Let $\mathsf{LZ}_{\mathsf{opt}}$ be the size in bits of an optimal LZ77 encoding of a string of length n over an alphabet of size $\sigma \geq 2$. Then, we have $\mathsf{LZ}_{\mathsf{opt}} = \Omega(\log n + z \log\log_\sigma z)$, where z is the number of phrases in the greedy LZ77 parsing of this string.*

Proof. Denote by $f_1 f_2 \cdots f_{z'}$ the LZ77 parsing corresponding to an optimal LZ77 encoding of the string under consideration. By the definition of the phrase encoders, we have $\mathsf{LZ}_{\mathsf{opt}} \geq \Omega(\sum_{i=1}^{z'} \log |f_i|)$. It follows from Lemma 3 that $\mathsf{LZ}_{\mathsf{opt}} \geq \Omega(\log(n-z'))$. Since, obviously, $\mathsf{LZ}_{\mathsf{opt}} \geq z'$, the latter implies $\mathsf{LZ}_{\mathsf{opt}} \geq \Omega(z' + \log(n-z')) \geq \Omega(\log n)$.

Denote by $f'_1 f'_2 \cdots f'_z$ the greedy LZ77 parsing of the same string. Let S be the set of all phrases in this parsing with lengths at least $\frac{1}{2}\log_\sigma z$. By Lemma 5, we have $|S| \geq$

$z - 2\sqrt{z} = \Theta(z)$. Consider a phrase $f' \in S$. Let $f_g, f_{g+1}, \ldots, f_h$ be all phrases in the parsing $f_1 f_2 \cdots f_{z'}$ that overlap with the phrase f'. Since $|f_g f_{g+1} \cdots f_h| \geq |f'|$, Lemma 3 implies that $(h-g)+\log|f_g|+\log|f_{g+1}|+\cdots+\log|f_h| \geq (h-g)+\log(|f'|-(h-g)) \geq \Omega(\log|f'|)$. Thus, the encodings of the phrases $f_g, f_{g+1}, \ldots, f_h$ all together occupy $\Omega(\log|f'|)$ bits. By Lemma 4, any phrase f_i of the parsing $f_1 \cdots f_{z'}$ overlaps with at most two phrases of the parsing $f'_1 \cdots f'_z$. Therefore, the encodings of all phrases $f_1, \ldots, f_{z'}$ occupy $\frac{1}{2}\Omega(\sum_{f' \in S} \log|f'|) \geq \Omega(|S| \log\log_\sigma z) = \Omega(z \log\log_\sigma z)$ overall bits. ◀

▶ **Theorem 7.** *Let z be the number of phrases in the greedy LZ77 parsing of a given string of length n drawn from an alphabet of size σ. Denote by $\mathsf{LZ_{gr}}$ and $\mathsf{LZ_{opt}}$ the sizes in bits of, respectively, the greedy and optimal LZ77 encodings of this string. Then, we have $\frac{\mathsf{LZ_{gr}}}{\mathsf{LZ_{opt}}} = O(\min\{z, \frac{\log n}{\log\log_\sigma z}\})$.*

Proof. By Lemmas 2 and 6, $\frac{\mathsf{LZ_{gr}}}{\mathsf{LZ_{opt}}} \leq \frac{O(z \log n)}{\Omega(\log n + z \log\log_\sigma z)} = O(\frac{z \log n}{\log n + z \log\log_\sigma z})$. Since $\frac{z \log n}{\log n + z \log\log_\sigma z} \leq \frac{z \log n}{\log n} = z$ and $\frac{z \log n}{\log n + z \log\log_\sigma z} \leq \frac{\log n}{\log\log_\sigma z}$, the result follows. ◀

▶ **Corollary 8.** *For constant alphabet, $\frac{\mathsf{LZ_{gr}}}{\mathsf{LZ_{opt}}} = O(\frac{\log n}{\log\log\log n})$.*

Proof. We have $\frac{\mathsf{LZ_{gr}}}{\mathsf{LZ_{opt}}} = O(\min\{z, \frac{\log n}{\log\log z}\})$ due to Theorem 7. The functions $z \mapsto z$ and $z \mapsto \frac{\log n}{\log\log z}$, respectively, increase and decrease as z grows. Therefore, the maximum of the function $\min\{z, \frac{\log n}{\log\log z}\}$ is reached when $z = \frac{\log n}{\log\log z}$. Solving this equation, we obtain $z = \Theta(\frac{\log n}{\log\log\log n})$, which proves the result. ◀

▶ Remark. To generalize the described results to nonclassical LZ77 parsings, one should use, instead of Lemma 1, the following straightforward lemma.

▶ **Lemma 9.** *Suppose that $s = f_1 f_2 \cdots f_z$ is the greedy nonclassical LZ77 parsing of a given string s; then, all the strings $f_i \cdot f_{i+1}[1]$, for $i \in [1..z-1]$, are distinct.*

The rest can be easily reconstructed by analogy.

4 Lower Bound

We now construct a series of example showing that, for several important cases, the upper bound given in Theorem 7 is tight. In particular, on constant alphabets, i.e., when $\sigma = O(1)$, Theorem 10 complements Theorem 7 showing that the bound $O(\min\{z, \frac{\log n}{\log\log z}\})$ is tight. Further, putting $z = \frac{\log n}{\log\log\log n}$ and $\sigma = 2$ in Theorem 10, we show that the upper bound given in Corollary 8 is tight.

▶ **Theorem 10.** *For any given integers $n > 1$, $\sigma \in [2..n]$, and $z \in [\sigma..\frac{n}{\log_\sigma n}]$, there is a string of length n over an alphabet of size σ such that the number of phrases in the greedy LZ77 parsing of this string is $\Theta(z)$ and the sizes $\mathsf{LZ_{gr}}$ and $\mathsf{LZ_{opt}}$ of, respectively, the greedy and optimal LZ77 encodings of this string are related as $\frac{\mathsf{LZ_{gr}}}{\mathsf{LZ_{opt}}} \geq \Omega(\min\{z, \frac{\log n}{\log\log_\sigma z + \log\sigma}\})$.*

Proof. If $\sigma \geq n/4$, then any LZ77 encoding of a string of length n containing σ distinct letters obviously occupies $\Theta(\sigma \log\sigma) = \Theta(n \log n)$ bits and, hence, the statement of the theorem, which degenerates to $\frac{\mathsf{LZ_{gr}}}{\mathsf{LZ_{opt}}} \geq \Omega(1)$, trivially holds. Assume that $\sigma < n/4$.

We first consider the case $\sigma \geq 3$ as it is simpler. Suppose that the alphabet is the set $[1..\sigma]$. Denote $b = 1$ and $\tau = \sigma - 1$ (b is a special letter-separator with small code and τ is the size of the set $[1..\sigma] \setminus \{b\} = [2..\sigma]$). Let m be the minimal integer such that $\tau^m \geq z$,

i.e., $m = \lceil \log_\tau z \rceil$. Note that $m = \Theta(\log_\sigma z)$. In [5] it is shown that all τ^m possible strings of length m over the alphabet $[2..\sigma]$ can be arranged in a sequence $s_1, s_2, \ldots, s_{\tau^m}$ (called a *τ-ary Gray code* [5, 13]) such that, for any $i \in [2..\tau^m]$, the strings s_{i-1} and s_i differ in exactly one position. Moreover, we can choose such sequence so that $s_{\tau^m} = a^m$, where a is an arbitrary letter from $[2..\sigma]$.

Let k and ℓ be positive integers such that $k < \tau^m$ and $\ell > m$. Our example is the following string (the numbers k and ℓ will be adjusted below so that $k = \Theta(z)$ and $\ell \geq \frac{1}{2}n$):

$$s = s_1 s_2 \cdots s_k \cdot a^\ell \cdot b s_1 b s_2 b \cdots s_k b.$$

Let us consider the greedy LZ77 parsing of s and the corresponding greedy LZ77 encoding. Since the letter b first occurs in the substring $a^\ell b$, the greedy construction procedure builds the parsing of $s_1 b s_2 b \cdots s_k b$ starting from the first position of this substring. Since $k < \tau^m$ and $s_{\tau^m} = a^m$, it follows from the definition of the sequence $s_1, \ldots, s_k$ that, for any $i \in [1..k]$, the longest prefix of the string $s_i b s_{i+1} b \cdots s_k b$ that has an earlier occurrence in s is s_i and this earlier occurrence is a substring of the prefix $s_1 s_2 \cdots s_k a^m$ of s. Therefore, the greedy algorithm decomposes the suffix $s_1 b s_2 b \cdots s_k b$ into k phrases $s_i b$, for $i \in [1..k]$. It is easy to see that each of these phrases is encoded in $\Omega(\log \ell)$ bits (this is the number of bits required to encode the distance between the phrase and its earlier occurrence). Hence, the size in bits of the greedy LZ77 encoding of s is $\mathsf{LZ}_{\mathsf{gr}} \geq \Omega(k \log \ell)$.

Now let us consider a better encoding of the same string s. For simplicity, we omit the description of the encoding of the prefix $s_1 s_2 \cdots s_k$ as it is very similar to the encoding of the suffix $s_1 b s_2 b \cdots s_k b$ discussed below. First, we parse the substring $a^\ell b$ into two phrases a and $a^{\ell-1}b$, which are encoded in $O(\log \ell + \log \sigma)$ bits (the referenced part $a^{\ell-1}$ of $a^{\ell-1}b$ is self-referential). Then, we encode the substring $s_1 b$ as in the greedy approach by one phrase taking $O(\log \ell)$ bits (recall that $\ell > m$ and $b = 1$ and, hence, the length $|s_1 b| = m+1$ and the letter b are encoded in $O(\log \ell)$ bits). Now we consecutively encode each substring $s_i b$, for $i \in [2..k]$, as follows. Suppose that the strings s_i and s_{i-1} differ at position j, i.e., $s_{i-1}[1..j-1] = s_i[1..j-1]$ and $s_{i-1}[j+1..m] = s_i[j+1..m]$. We decompose $s_i b$ into two phrases $s_i[1..j]$ and $s_i[j+1..m]b$. Since the strings $s_i[1..j-1]$ and $s_i[j+1..m]$ both are substrings of the string s_{i-1} and have length $O(m)$, the encoding of the produced two phrases occupies $O(\log m + \log \sigma)$ bits. Hence, the whole suffix $s_1 b s_2 b \cdots s_k b$ can be encoded in $O(k \log m + k \log \sigma)$ bits; the prefix $s_1 s_2 \cdots s_k$ can be encoded similarly in $O(k \log m + k \log \sigma)$ bits. Thus, we obtain an encoding of the string s that occupies $O(\log \ell + k \log m + k \log \sigma)$ bits. Therefore, the size in bits of the optimal LZ77 encoding of s is $\mathsf{LZ}_{\mathsf{opt}} = O(\log \ell + k \log m + k \log \sigma)$.

Recall that $m = \Theta(\log_\sigma z)$. Combining the estimations on $\mathsf{LZ}_{\mathsf{gr}}$ and $\mathsf{LZ}_{\mathsf{opt}}$, we obtain $\frac{\mathsf{LZ}_{\mathsf{gr}}}{\mathsf{LZ}_{\mathsf{opt}}} \geq \frac{\Omega(k \log \ell)}{O(\log \ell + k(\log m + \log \sigma))} \geq \Omega(\frac{k \log \ell}{\log \ell + k(\log \log_\sigma z + \log \sigma)})$. Since $\frac{k \log \ell}{\log \ell + k(\log \log_\sigma z + \log \sigma)} \geq \frac{k \log \ell}{2 \cdot \max\{\log \ell, k(\log \log_\sigma z + \log \sigma)\}} = \frac{1}{2} \min\{k, \frac{\log \ell}{\log \log_\sigma z + \log \sigma}\}$, we obtain $\frac{\mathsf{LZ}_{\mathsf{gr}}}{\mathsf{LZ}_{\mathsf{opt}}} \geq \Omega(\min\{k, \frac{\log \ell}{\log \log_\sigma z + \log \sigma}\})$. Note that the number of phrases in the greedy LZ77 parsing of s is $\Theta(k)$ and $|s| = \ell + 1 + k(2m+1)$. We put $\ell = n - k(2m+1) - 1$ so that $|s| = n$. Since $z \in [2..\frac{n}{\log_\sigma n}]$ and $m = \Theta(\log_\sigma z)$, we have $k(2m+1) \leq O(n)$ if $k = \Theta(z)$. Then, it is straightforward that the parameter k can be chosen so that $k = \Theta(z)$ and $\ell = n - k(2m+1) - 1 \geq \frac{1}{2}n$. Hence, we derive $\frac{\mathsf{LZ}_{\mathsf{gr}}}{\mathsf{LZ}_{\mathsf{opt}}} \geq \Omega(\min\{z, \frac{\log n}{\log \log_\sigma z + \log \sigma}\})$. (If not all letters of the alphabet $[1..\sigma]$ indeed occur in the constructed string, we append all unused letters to the end of s and reduce ℓ appropriately; as $\sigma < n/4$, we have $\ell \geq \frac{1}{4}n$ in the end.)

Now assume that $\sigma = 2$. Let $\{0, 1\}$ be the alphabet. Similarly to the above analysis, we fix a sequence $s_1, \ldots, s_{2^m}$ of all binary strings of length $m = \lceil \log z \rceil$ such that, for $i \in [2..2^m]$, s_{i-1} and s_i differ in exactly one position, and we choose two parameters $\ell > 4m$ and $k < 2^m$,

which will be adjusted later so that $\ell \geq \frac{1}{2}n$ and $k = \Theta(z)$. It is well known that one can fix the sequence $s_1, \ldots, s_{2^m}$ so that $s_{2^m} = 0^m$. Our example is defined as follows:

$$s = s_1 0^m 1 s_2 0^m 1 \cdots s_k 0^m 1 0^\ell 1 s_1 0^m 1 c_1 s_2 0^m 1 c_2 \cdots s_k 0^m 1 c_k,$$

where $c_k = 1$ and, for $i \in [1..k-1]$, $c_i = 0$ if $s_{i+1}[1] = 1$, and $c_i = 1$ otherwise.

Since, for any $i \in [1..k]$, $s_i \neq 0^m$ (as $s_i = 0^m$ iff $i = 2^m$, and $k < 2^m$) and $\ell > 4m$, the greedy LZ77 parser necessarily makes a phrase that is a suffix of the substring $0^\ell 1$ and, then, parses the suffix $s_1 0^m 1 c_1 s_2 0^m 1 c_2 \cdots s_k 0^m 1 c_k$ from the first position. It is straightforward that, for any $i \in [1..k]$, the string $0^m 1$ has only one occurrence in the strings $1 s_i 0^m 1$ and $1 c_{i-1} s_i 0^m 1$ (for $i > 1$). Therefore, for any $i \in [1..k]$, the string $s_i 0^m 1$ has only one occurrence in the prefix $s_1 0^m 1 s_2 0^m 1 \cdots s_k 0^m 1$ and the string $s_i 0^m 1 c_i$ has only one occurrence in the whole string s. Then, the greedy parser parses the suffix $s_1 0^m 1 c_1 s_2 0^m 1 c_2 \cdots s_k 0^m 1 c_k$ into k phrases $s_i 0^m 1 c_i$, for $i \in [1..k]$. This parsing produces an encoding of size $\Omega(k \log \ell)$ bits. At the same time, there is an LZ77 encoding for s of size $O(\log \ell + k \log m)$ bits. The further analysis is very similar to the analysis of the case $\sigma \geq 3$: we put $\ell = n - k(4m + 3) - 1$ so that $|s| = n$, and we adjust k so that $k = \Theta(z)$ and $\ell \geq \frac{1}{2}n$, which is possible because $m \leq \log z + 1$ and $z \leq \frac{n}{\log n}$. We omit the details as they are analogous. ◀

▶ Remark. The condition $\sigma \leq z \leq \frac{n}{\log_\sigma n}$ from Theorem 10 is justified by the following observations. First, it is obvious that any LZ77 parsing has at least σ phrases and, hence, the inequality $\sigma \leq z$ holds. Secondly, by Lemma 5, at least $z - 2\sqrt{z}$ phrases in the greedy LZ77 parsing have length at least $\frac{1}{2}\log_\sigma z$, where z is the total number of phrases; hence, we obtain $z \log_\sigma z \leq O(n)$ and, solving this inequality, $z = O(\frac{n}{\log_\sigma n})$, which justifies the condition $z \leq \frac{n}{\log_\sigma n}$.

▶ Remark. Let us sketch the way in which the constructions from the proof of Theorem 10 can be adapted to nonclassical LZ77 encodings. For the case $\sigma \geq 3$, the corresponding string is as follows (the notation is from the proof of Theorem 10):

$$s = b s_1 b s_2 \cdots b s_k b \cdot a^\ell \cdot b s_1 b b s_2 b b \cdots b b s_k b.$$

The suffix $b s_1 b b s_2 b b \cdots b b s_k b$ of this string is greedily parsed into the phrases $b s_i b$, for $i \in [1..k]$. For the case $\sigma = 2$, the corresponding string is as follows:

$$s = 10 s_1 \alpha 10 s_2 \alpha 1 \cdots 10 s_k \alpha \cdot 0^\ell \cdot 10 s_1 \alpha 0 s_2 \alpha 0 \cdots 0 s_k \alpha 0,$$

where $\alpha = 0^{m+1}1$. The suffix $10 s_1 \alpha 0 s_2 \alpha 0 \cdots 0 s_k \alpha 0$ of s is greedily parsed into the phrases $10 s_1 \alpha$ and $0 s_i \alpha$, for $i \in [2..k]$. We omit the detailed analysis as it is analogous to the analysis in the proof of Theorem 10.

5 Arbitrary Alphabets

The following corollary shows that, in the case of "non-extremely compressible" string ($z \geq 2^{\log^\epsilon n}$) over a polylogarithmic alphabet ($\sigma \leq \log^{O(1)} n$), which is arguably the most important case for practice, the upper and lower bounds from Theorems 7 and 10 degenerate to $\Theta(\frac{\log n}{\log \log n})$ and, hence, are tight. (Note that $2^{\log^\epsilon n} = o(n^\delta)$ for any fixed constants $\epsilon \in (0, 1)$ and $\delta \in (0, 1)$.)

▶ **Corollary 11.** *Let z be the number of phrases in the greedy LZ77 parsing of a given string of length n drawn from an alphabet of size σ. Suppose that $\sigma \leq \log^{O(1)} n$ and $z \geq 2^{\log^\epsilon n}$, for a fixed constant $\epsilon \in (0, 1)$. Denote by $\mathsf{LZ_{gr}}$ and $\mathsf{LZ_{opt}}$ the sizes in bits of, respectively, the greedy and optimal LZ77 encodings of this string. Then, we have $\frac{\mathsf{LZ_{gr}}}{\mathsf{LZ_{opt}}} \leq O(\frac{\log n}{\log \log n})$ and this upper bound is tight.*

Proof. The result follows from Theorems 7 and 10 since $\log\log n \geq \log\log_\sigma z \geq \log \frac{\log^\epsilon n}{O(\log\log n)} = \Theta(\log\log n)$. ◀

Now let us consider bounds on the ratio $\frac{\mathsf{LZ}_{\mathsf{gr}}}{\mathsf{LZ}_{\mathsf{opt}}}$ that are independent of the parameters z and σ.

In [11] it was proved that $O(\log n)$ is an upper bound on the ratio $\frac{\mathsf{LZ}_{\mathsf{gr}}}{\mathsf{LZ}_{\mathsf{opt}}}$. It turns out that this bound is tight on sufficiently large non-constant alphabets. Precisely, a series of examples on which $\frac{\mathsf{LZ}_{\mathsf{gr}}}{\mathsf{LZ}_{\mathsf{opt}}} = \Omega(\log n)$ can be constructed on an alphabet of size $O(\log n)$. Therefore, the upper bound $O(\frac{\log n}{\log\log\log n})$ on the ratio $\frac{\mathsf{LZ}_{\mathsf{gr}}}{\mathsf{LZ}_{\mathsf{opt}}}$, which, by Corollary 8, holds for constant alphabets and is tight, does not hold, in general, even for alphabets of logarithmic size. In examples showing this, we use the following well-known combinatorial structure.

A *Steiner system* $S(t,k,n)$ is a set S of size n and a family of k-element subsets of S, called *blocks*, such that each subset of S of size t is contained in exactly one block. We are particularly interested in the Steiner systems $S(2, 2^{2^{i-1}}, 2^{2^i})$, which can be constructed for any positive integers i (the structure is realized on a finite affine plane of order $2^{2^{i-1}}$ and the blocks are lines in the plane; see [6]). It is well known that the number of blocks in the Steiner system $S(2, 2^{2^{i-1}}, 2^{2^i})$ is $\binom{2^{2^i}}{2}/\binom{2^{2^{i-1}}}{2}$.

▶ **Theorem 12.** *For any integer $n > 1$, there is a string of length n over an alphabet of size $O(\log n)$ such that the sizes $\mathsf{LZ}_{\mathsf{gr}}$ and $\mathsf{LZ}_{\mathsf{opt}}$ of, respectively, the greedy and optimal LZ77 encodings of this string are related as $\frac{\mathsf{LZ}_{\mathsf{gr}}}{\mathsf{LZ}_{\mathsf{opt}}} \geq \Omega(\log n)$.*

Proof. Let us first discuss a high-level idea of our construction. Consider the following string:

$$t \cdot b_1 c b'_1 \cdot b_2 c b'_2 \cdots b_k c b'_k \cdot c^{\Theta(n)} \cdot t d \cdot b_1 c b'_1 d \cdot b_2 c b'_2 d \cdots b_k c b'_k d,$$

where $t = a_1 a_2 \cdots a_{\sigma-2}$ is a string consisting of $\sigma-2$ distinct letters, the sets $\{b_i, b'_i\}$ run through all $k = \binom{\sigma-2}{2}$ two-element subsets of the set $\{a_1, a_2, \ldots, a_{\sigma-2}\}$, and c and d are two special letters with constant codes (say, 0 and 1) that do not occur in t. The greedy LZ77 parser parses the suffix $b_1 c b'_1 d \cdot b_2 c b'_2 d \cdots b_k c b'_k d$ into phrases $b_i c b'_i d$ encoded by references to the substrings $b_i c b'_i$ of the prefix $t \cdot b_1 c b'_1 \cdot b_2 c b'_2 \cdots b_k c b'_k$. Each such reference takes $\Omega(\log n)$ bits and, therefore, the greedy encoding occupies $\Omega(\binom{\sigma-2}{2}\log n) = \Omega(\sigma^2 \log n)$ bits.

Obviously, any LZ77 encoding spends $\Theta(\log n)$ bits to encode the substring $c^{\Theta(n)}$. If we were able to encode the prefix and the suffix surrounding the substring $c^{\Theta(n)}$ in $O(\sigma^2)$ bits, then we would obtain $\frac{\mathsf{LZ}_{\mathsf{gr}}}{\mathsf{LZ}_{\mathsf{opt}}} \geq \Omega(\frac{\sigma^2 \log n}{\sigma^2 + \log n}) = \Omega(\frac{\sigma^2 \log n}{\max\{\sigma^2, \log n\}}) = \Omega(\min\{\log n, \sigma^2\})$, which is $\Omega(\log n)$ for $\sigma = \Omega(\sqrt{\log n})$. Unfortunately, it seems that the best encoding that one can find for the suffix $b_1 c b'_1 d \cdot b_2 c b'_2 d \cdots b_k c b'_k d$ parses each substring $b_i c b'_i d$ into two phrases $b_i c$ and $b'_i c$, encoding each of them by a reference to a letter in $t = a_1 a_2 \cdots a_{\sigma-2}$, thus spending $\Theta(\binom{\sigma-2}{2}\log\binom{\sigma-2}{2}) = \Theta(\sigma^2 \log\sigma)$ bits for the whole suffix, which is larger than $\Theta(\sigma^2)$ by the factor $\log\sigma$. To address this issue, we construct a more sophisticated string equipped with additional "infrastructure" that helps to "deliver" cheaply letters from a "dictionary" substring (like t) to the places where these letters are used. Let us formalize this intuition.

Choose the minimal positive integer x such that $2^{2^x} > \sqrt{\log n}$. The alphabet for our example will consist of two special letters c and d with codes 0 and 1, and of the set A of 2^{2^x} letters with codes larger than 1. Obviously, the alphabet size $\sigma = 2^{2^x} + 2$ is at most $\log n + 2$.

Let us assign to each subset S of A such that $|S| = 2^{2^i}$, for some $i \in [1..x]$, a Steiner system $S(2, 2^{2^{i-1}}, 2^{2^i})$ with the set of blocks denoted by B_S. Denote by q a mapping that maps every such S to a string $q(S) = a_{j_1} d a_{j_2} d \cdots a_{j_{|S|}} d$, where $a_{j_1}, a_{j_2}, \ldots, a_{j_{|S|}}$ are all letters

from S in an arbitrarily chosen order. The basic building elements for our string are defined recursively as follows.

$$r(S) = q(S)\prod_{B\in B_S} r(B) \quad \text{if } |S| > 2,$$
$$r(S) = bcb'cbcb'dd \quad \text{if } S = \{b, b'\} \text{ for distinct letters } b, b'.$$

Analogously, we define:

$$r'(S) = q(S)\prod_{B\in B_S} r(B) \quad \text{if } |S| > 2,$$
$$r'(S) = bcb'cbcb'dc \quad \text{if } S = \{b, b'\} \text{ for distinct letters } b, b'.$$

To break ties on the lowest levels of recursion where $|S| = 2$, we assume that b is the letter from S with the smallest code.

Our string on which $\frac{\mathsf{LZ}_{\mathsf{gr}}}{\mathsf{LZ}_{\mathsf{opt}}} \geq \Omega(\log n)$ is $s = r'(A)c^\ell r(A)$, where ℓ is chosen so that $\ell = \Theta(n)$ (see the text below, where we discuss the lengths of $r(S)$ and $r'(S)$). Let us first show that the greedy LZ77 encoding of this string has size $\Omega(\sigma^2 \log n)$ bits.

By the definition of Steiner systems, for any subset $S \subseteq A$ of size 2^{2^i}, each pair $\{b, b'\}$ of distinct letters from S is contained in exactly one block (of size $2^{2^{i-1}}$) from B_S. Then, it is straightforward that any given pair $\{b, b'\}$ of distinct letters from A occurs exactly once as a parameter of r on the lowest level of the recursion $r(A)$. An analogous claim holds for $r'(A)$. Hence, the string $bcb'cbcb'd$ (we assume that the code of b is smaller than the code of b') occurs in s exactly twice: in the prefix $r'(A)$ and in the suffix $r(A)$. Further, it is easy to see that the string bcb' occurs in s only as a substring of $bcb'cbcb'd$. By a straightforward case analysis, one can show that this implies that the greedy LZ77 parsing of s has a phrase f containing the substring $bcb'dd$ of $r(A)$: f either is a phrase starting at one of the first five positions of $bcb'cbcb'dd$ (greedily "eating" the remaining part) or is a phrase containing the prefix $bcb'cb$ of $bcb'cbcb'dd$ (the part $bcb'c$ can be copied only from $bcb'cbcb'dc$ in $r'(A)$ and, thus, again f greedily "eats" the remaining part). The encoding of f copies the part $bcb'd$ from the substring $bcb'd$ of $r'(A)$ by reference, thus spending $\Omega(\log \ell) = \Omega(\log n)$ bits. Since the two occurrences of $bcb'cbcb'd$ in s are followed by distinct letters (c in $r'(A)$ and d in $r(A)$), the string $bcb'dd$ must be a suffix of f. Hence, there is a one-to-one correspondence between the pairs $\{b, b'\}$ of distinct letters from A and the phrases containing the substrings $bcb'dd$. Therefore, the greedy LZ77 encoding of s occupies $\Omega(\binom{|A|}{2}\log n) = \Omega(\sigma^2 \log n)$ bits.

Now it remains to show that there is an LZ77 encoding of the string s that occupies $O(\sigma^2 + \log n)$ bits. This will imply that $\frac{\mathsf{LZ}_{\mathsf{gr}}}{\mathsf{LZ}_{\mathsf{opt}}} \geq \frac{\Omega(\sigma^2 \log n)}{O(\sigma^2+\log n)} \geq \Omega(\min\{\log n, \sigma^2\})$, which is $\Omega(\log n)$ since, by construction, $\sigma > \sqrt{\log n}$.

We decompose the substring c^ℓ of $s = r'(A)c^\ell r(A)$ into two phrases c and $c^{\ell-1}$, encoding these phrases in $O(\log n)$ bits. All other phrases in our parsing will have length either one or two. For simplicity of the exposition, we consider only encoding of the suffix $r(A)$; the encoding for $r'(A)$ is analogous and occupies asymptotically the same space.

By definition, $q(A)$ is a prefix of $r(A)$. The string $q(A)$ serves as a "dictionary" of letters similar to the string t in the preliminary example. We encode each letter of $q(A)$ as a phrase of length one, thus spending $O(\sigma \log \sigma)$ bits. These are the only "heavy" phrases of length one in our encoding of $r(A)$: all other phrases of length one will be either c or d, the letters with codes 0 and 1, which can be encoded in $O(1)$ bits. All phrases of length two will have the form either ac or ad, where $a \in A$; thus, the "heavy" part of the encoding of such phrases of length two is an $O(\log \delta)$-bit encoding of the distance δ to an occurrence of a preceding this phrase.

Let us consider a substring $r(S) = q(S)\prod_{B\in B_S} r(B)$ of $r(A)$, where $S \subseteq A$ is a set of size 2^{2^i} that occurs in the expansion of the recursion $r(A)$. Suppose that $i > 1$. Then, each

substring $r(B)$, for $B \in B_S$, has a prefix $q(B) = a_1 d a_2 d \cdots a_{|B|} d$, where $a_1, a_2, \ldots, a_{|B|}$ are members of B. We parse $q(B)$ into phrases $a_1 d, a_2 d, \ldots, a_{|B|} d$, encoding each phrase $a_i d$ by a reference to the letter a_i of the prefix $q(S)$ of $r(S)$. Suppose that $i = 1$. Then, each block $B \in B_S$ is just a pair $\{b, b'\}$ of distinct letters from S, and $r(B) = bcb'cbcb'dd$. We parse $r(B)$ into phrases $bc, b'c, bc, b'd, d$, encoding each phrase of length two by a reference to a letter from the prefix $q(S)$ of the string $r(S)$.

Denote by $E(i)$ the maximum size in bits of the encoding for the suffix $\prod_{B \in B_S} r(B)$ of some string $r(S)$, among all subsets $S \subseteq A$ such that $|S| = 2^{2^i}$. Then, $E(i)$ can be expressed by the following recursion (recall that $|B_S| = \binom{2^{2^i}}{2} / \binom{2^{2^{i-1}}}{2}$):

$$
\begin{aligned}
&E(i) \le \left(\binom{2^{2^i}}{2} / \binom{2^{2^{i-1}}}{2} \right) (2^{2^{i-1}} \alpha \log L(i) + E(i-1)), \quad \text{for } i > 1,\\
&E(1) \le \tbinom{4}{2} (4\alpha \log L(1) + \alpha),
\end{aligned}
$$

where $L(i)$ denotes the length of the string $r(S)$ (obviously, L depends only on the size 2^{2^i} of S) and α is a positive constant that depends on the chosen phrase encoder. Consider the prefix $q(B)$ of a substring $r(B)$ of $r(S)$, where $B \in B_S$ and $|B| > 2$. Each phrase ad from the parsing of $q(B)$ is encoded in $O(\log \delta)$ bits, where δ is the distance to the letter a from the prefix $q(S)$ of $r(S)$. Obviously, we have $\delta < L(i)$. Therefore, choosing an appropriate constant $\alpha > 0$, we can estimate the number of bits required to encode all $2^{2^{i-1}}$ phrases from the parsing of $q(B)$ as $2^{2^{i-1}} \alpha \log L(i)$; hence, the expression for $E(i)$ with $i \ne 1$. Analogously, the size in bits of the encoding for $bcb'cbcb'dd$ can be estimated as $4\alpha \log L(1) + \alpha$; hence, the expression for $E(1)$.

Thus, the whole encoding of the string s requires $O(\log n + \sigma \log \sigma + E(x))$ bits. It remains to show that $E(x) \le O(\sigma^2)$. Before finding a closed form for $E(i)$, let us consider $L(i)$, which can be expressed by the following recursion:

$$
\begin{aligned}
&L(i) = 2 \cdot 2^{2^i} + \left(\binom{2^{2^i}}{2} / \binom{2^{2^{i-1}}}{2} \right) L(i-1), \quad \text{for } i > 0,\\
&L(0) = 9.
\end{aligned}
$$

Here, $L(0) = |bcb'cbcb'dd| = 9$. Let us find a closed form for $L(i)$. Note that $2^{2^z} / \binom{2^{2^z}}{2} = \frac{2}{2^{2^z}-1}$ for any integer $z \ge 0$. Expanding the recursion for $L(i)$, we obtain:

$$
\begin{aligned}
&L(i) = 2 \cdot 2^{2^i} + \frac{\binom{2^{2^i}}{2}}{\binom{2^{2^{i-1}}}{2}} L(i-1)\\
&= 2^{2^i+1} + \frac{\binom{2^{2^i}}{2}}{\binom{2^{2^{i-1}}}{2}} \left(2 \cdot 2^{2^{i-1}} + \frac{\binom{2^{2^{i-1}}}{2}}{\binom{2^{2^{i-2}}}{2}} L(i-2) \right)\\
&= 2^{2^i+1} + \frac{4 \cdot \binom{2^{2^i}}{2}}{2^{2^{i-1}}-1} + \frac{\binom{2^{2^i}}{2}}{\binom{2^{2^{i-2}}}{2}} L(i-2)\\
&= 2^{2^i+1} + \left(\frac{4 \cdot \binom{2^{2^i}}{2}}{2^{2^{i-1}}-1} + \frac{4 \cdot \binom{2^{2^i}}{2}}{2^{2^{i-2}}-1} + \cdots + \frac{4 \cdot \binom{2^{2^i}}{2}}{2^{2^1}-1} \right) + 9 \cdot \tbinom{2^{2^i}}{2}\\
&= 2^{2^i+1} + \tbinom{2^{2^i}}{2} \left(\frac{4}{2^{2^{i-1}}-1} + \frac{4}{2^{2^{i-2}}-1} + \cdots + \frac{4}{2^{2^1}-1} + 9 \right).
\end{aligned}
$$

The term $9 \cdot \binom{2^{2^i}}{2}$ appears because of the last level of the recursion $L(i)$. Now it is easy to see that $L(i) \le \beta \cdot \binom{2^{2^i}}{2}$ for a constant $\beta > 0$. In particular, we obtain $|r(A)| = |r'(A)| = L(x) \le \beta \cdot \binom{2^{2^x}}{2} \le O(\sigma^2)$ (recall that $\sigma = 2^{2^x} + 2$). Since, as it was noted above, $\sigma \le \log n + 2$, we obtain $L(x) \le O(\log^2 n)$. Hence, for large enough n, we have

$\ell = n - |r(A)| - |r'(A)| = n - 2L(x) = n - O(\log^2 n) \geq \frac{1}{2}n$, i.e., $\ell = \Theta(n)$, as it was announced above. Let us similarly estimate $E(i)$. Denote $\gamma_i = \alpha \log L(i)$ for brevity.

$$
\begin{aligned}
E(i) &\leq \frac{\binom{2^{2^i}}{2}}{\binom{2^{2^{i-1}}}{2}}(2^{2^{i-1}}\gamma_i + E(i-1)) \\
&= \frac{2\cdot\binom{2^{2^i}}{2}}{2^{2^{i-1}}-1}\gamma_i + \frac{\binom{2^{2^i}}{2}}{\binom{2^{2^{i-1}}}{2}}E(i-1) \\
&= \frac{2\cdot\binom{2^{2^i}}{2}}{2^{2^{i-1}}-1}\gamma_i + \frac{\binom{2^{2^i}}{2}}{\binom{2^{2^{i-1}}}{2}}\left(\frac{\binom{2^{2^{i-1}}}{2}}{\binom{2^{2^{i-2}}}{2}}(2^{2^{i-2}}\gamma_{i-1} + E(i-2))\right) \\
&= \frac{2\cdot\binom{2^{2^i}}{2}}{2^{2^{i-1}}-1}\gamma_i + \frac{\binom{2^{2^i}}{2}}{\binom{2^{2^{i-2}}}{2}}2^{2^{i-2}}\gamma_{i-1} + \frac{\binom{2^{2^i}}{2}}{\binom{2^{2^{i-2}}}{2}}E(i-2) \\
&= \frac{2\cdot\binom{2^{2^i}}{2}}{2^{2^{i-1}}-1}\gamma_i + \frac{2\cdot\binom{2^{2^i}}{2}}{2^{2^{i-2}}-1}\gamma_{i-1} + \frac{\binom{2^{2^i}}{2}}{\binom{2^{2^{i-2}}}{2}}E(i-2) \\
&= \frac{2\cdot\binom{2^{2^i}}{2}}{2^{2^{i-1}}-1}\gamma_i + \frac{2\cdot\binom{2^{2^i}}{2}}{2^{2^{i-2}}-1}\gamma_{i-1} + \cdots + \frac{2\cdot\binom{2^{2^i}}{2}}{2^{2^1}-1}\gamma_2 + \binom{2^{2^i}}{2}(4\gamma_1+\alpha) \\
&= 2\cdot\binom{2^{2^i}}{2}\left(\frac{\gamma_i}{2^{2^{i-1}}-1} + \frac{\gamma_{i-1}}{2^{2^{i-2}}-1} + \cdots + \frac{\gamma_2}{2^{2^1}-1} + 2\gamma_1 + \frac{\alpha}{2}\right).
\end{aligned}
$$

The term $\binom{2^{2^i}}{2}(4\gamma_1 + \alpha)$ appear because of the last level of the recursion $E(i)$. Note that $\gamma_i = \alpha \log L(i) \leq \alpha \log(\beta \cdot \binom{2^{2^i}}{2}) = O(2^i)$. It is well known that $\sum_{k=0}^{\infty} \frac{2^k}{2^{2^k}-1} = O(1)$. Therefore, $E(i)$ can be estimated as $O(\binom{2^{2^i}}{2})$. Thus, we obtain $E(x) \leq O(\binom{2^{2^x}}{2})$, which is $O(\sigma^2)$ since $\sigma = 2^{2^x} + 2$. ◀

▶ Remark. For nonclassical LZ77 encodings, we can use exactly the same example as in the proof of Theorem 12. In this case, the substrings $q(B) = a_1 d a_2 d \cdots a_{|B|} d$ and $bcb'cbcb'dd$ of each string $r(S)$ are parsed into one-letter phrases: the phrases c and d are encoded in $O(1)$ bits using the codes of these letters, and the phrases $a_1, a_2, \ldots, a_{|B|}, b, b'$ are encoded using references to letters of the prefix $q(S)$ of $r(S)$. The analysis of the size of thus obtained encoding is analogous.

6 Concluding Remarks

The upper and lower bounds $O(\min\{z, \frac{\log n}{\log\log_\sigma z}\})$ and $\Omega(\min\{z, \frac{\log n}{\log\log_\sigma z + \log\sigma}\})$, established in Theorems 7 and 10, completely solve the problem for the case of constant alphabets and for some cases of arbitrary alphabets. But the general case of arbitrary alphabets with bounds expressed in terms of the parameters n, z, σ remains open (see Table 1 in the introduction). Note that the examples constructed in the proof of Theorem 12 to show that $\frac{\mathsf{LZ}_{\mathsf{gr}}}{\mathsf{LZ}_{\mathsf{opt}}} \geq \Omega(\log n)$ are extremely compressible strings with $z = O(\log^2 n)$ and it is not clear whether the upper bound $\frac{\mathsf{LZ}_{\mathsf{gr}}}{\mathsf{LZ}_{\mathsf{opt}}} \leq O(\log n)$ remains tight if we consider "non-extremely compressible" strings (but not necessarily on polylogarithmic alphabets).

It is interesting to consider other encoders for LZ77. Many practical compressors utilize a type of phrase encoders that is strikingly different from ours: such encoders use entropy compression as a component. DEFLATE and LZMA are important examples of compression schemes using such techniques. This is a major open problem to formalize these schemes and to conduct a similar theoretical analysis of the efficiency of the popular greedy approach.

References

1. Amihood Amir, Gad M. Landau, and Esko Ukkonen. Online timestamped text indexing. *Inf. Process. Lett.*, 82(5):253–259, 2002. doi:10.1016/S0020-0190(01)00275-7.
2. Djamal Belazzougui and Simon J. Puglisi. Range predecessor and lempel-ziv parsing. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 2053–2071. SIAM, 2016. doi:10.1137/1.9781611974331.ch143.
3. Philip Bille, Patrick Hagge Cording, Johannes Fischer, and Inge Li Gørtz. Lempel-ziv compression in a sliding window. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, volume 78 of *LIPIcs*, pages 15:1–15:11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.15.
4. Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Trans. Information Theory*, 51(7):2554–2576, 2005. doi:10.1109/TIT.2005.850116.
5. Martin Cohn. Affine m-ary gray codes. *Information and Control*, 6(1):70–78, 1963. doi:10.1016/S0019-9958(63)90119-0.
6. C. J. Colbourn and J. H. Dinitz. *Handbook of combinatorial designs*. CRC press, 2006.
7. Maxime Crochemore, Laura Giambruno, Alessio Langiu, Filippo Mignosi, and Antonio Restivo. Dictionary-symbolwise flexible parsing. *J. Discrete Algorithms*, 14:74–90, 2012. doi:10.1016/j.jda.2011.12.021.
8. Maxime Crochemore, Alessio Langiu, and Filippo Mignosi. The rightmost equal-cost position problem. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2013 Data Compression Conference, DCC 2013, Snowbird, UT, USA, March 20-22, 2013*, pages 421–430. IEEE, 2013. doi:10.1109/DCC.2013.50.
9. Maxime Crochemore, Alessio Langiu, and Filippo Mignosi. Note on the greedy parsing optimality for dictionary-based text compression. *Theor. Comput. Sci.*, 525:55–59, 2014. doi:10.1016/j.tcs.2014.01.013.
10. Peter Elias. Universal codeword sets and representations of the integers. *IEEE Trans. Information Theory*, 21(2):194–203, 1975. doi:10.1109/TIT.1975.1055349.
11. Paolo Ferragina, Igor Nitto, and Rossano Venturini. On the bit-complexity of lempel-ziv compression. *SIAM J. Comput.*, 42(4):1521–1541, 2013. doi:10.1137/120869511.
12. Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. Lz77-based self-indexing with faster pattern matching. In Alberto Pardo and Alfredo Viola, editors, *LATIN 2014: Theoretical Informatics - 11th Latin American Symposium, Montevideo, Uruguay, March 31 - April 4, 2014. Proceedings*, volume 8392 of *Lecture Notes in Computer Science*, pages 731–742. Springer, 2014. doi:10.1007/978-3-642-54423-1_63.
13. F. Gray. Pulse code communication, 1953. US Patent 2,632,058.
14. S. R. Kosaraju and G. Manzini. Some entropic bounds for Lempel–Ziv algorithms. In *DCC 1997*, page 446. IEEE, 1997. doi:10.1109/DCC.1997.582106.
15. Sebastian Kreft and Gonzalo Navarro. On compressing and indexing repetitive sequences. *Theor. Comput. Sci.*, 483:115–133, 2013. doi:10.1016/j.tcs.2012.02.006.
16. N. Jesper Larsson. Most recent match queries in on-line suffix trees. In Alexander S. Kulikov, Sergei O. Kuznetsov, and Pavel A. Pevzner, editors, *Combinatorial Pattern Matching - 25th Annual Symposium, CPM 2014, Moscow, Russia, June 16-18, 2014. Proceedings*, volume 8486 of *Lecture Notes in Computer Science*, pages 252–261. Springer, 2014. doi:10.1007/978-3-319-07566-2_26.
17. V. I. Levenshtein. On the redundancy and delay of decodable coding of natural numbers. *Systems Theory Research*, 20:149–155, 1968.

18 Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1):2, 2007. doi:10.1145/1216370.1216372.

19 N. Rajpoot and C. Sahinalp. Dictionary-based data compression. *in Handbook of Lossless Data Compression*, pages 153–167, 2002.

20 Wojciech Rytter. Application of lempel-ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003. doi:10.1016/S0304-3975(02)00777-6.

21 James A. Storer and Thomas G. Szymanski. Data compression via textural substitution. *J. ACM*, 29(4):928–951, 1982. doi:10.1145/322344.322346.

22 A. D. Wyner and J. Ziv. The sliding-window Lempel–Ziv algorithm is asymptotically optimal. *Proceedings of the IEEE*, 82(6):872–877, 1994. doi:10.1109/5.286191.

23 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Information Theory*, 23(3):337–343, 1977. doi:10.1109/TIT.1977.1055714.

24 Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Information Theory*, 24(5):530–536, 1978. doi:10.1109/TIT.1978.1055934.

Width of Non-Deterministic Automata

Denis Kuperberg
CNRS, ENS de Lyon, LIP, France

Anirban Majumdar
Chennai Mathematical Institute, India

Abstract

We introduce a measure called width, quantifying the amount of nondeterminism in automata. Width generalises the notion of good-for-games (GFG) automata, that correspond to NFAs of width 1, and where an accepting run can be built on-the-fly on any accepted input. We describe an incremental determinisation construction on NFAs, which can be more efficient than the full powerset determinisation, depending on the width of the input NFA. This construction can be generalised to infinite words, and is particularly well-suited to coBüchi automata in this context. For coBüchi automata, this procedure can be used to compute either a deterministic automaton or a GFG one, and it is algorithmically more efficient in this last case. We show this fact by proving that checking whether a coBüchi automaton is determinisable by pruning is NP-complete. On finite or infinite words, we show that computing the width of an automaton is PSPACE-hard.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects, Theory of computation → Regular languages

Keywords and phrases Width, Non-deterministic Automata, Determinisation, Good-for-games, Complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.47

Funding This work was supported by the grant PALSE Impulsion.

Acknowledgements We thank Nathalie Bertrand for helpful discussions.

1 Introduction

Determinisation of non-deterministic automata (NFAs) is one of the cornerstone problems of automata theory, with countless applications in verification. There is a very active field of research for optimizing or approximating determinisation, or circumventing it in contexts like inclusion of NFA or Church Synthesis. Indeed, determinisation is a costly operation, as the state space blow-up is in $O(2^n)$ on finite words, $O(3^n)$ for coBüchi automata [16], and $2^{O(n \log(n))}$ for Büchi automata [17].

If $\mathcal{A}$ and $\mathcal{B}$ are NFAs, the classical way of checking the inclusion $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is to determinise $\mathcal{B}$, complement it, and test emptiness of $L(A) \cap \overline{L(B)}$. To circumvent a full determinisation, the recent algorithm from [3] proved to be very efficient, as it is likely to explore only a part of the powerset construction. Other approaches use simulation games to approximate inclusion at a cheaper cost, see for instance [8].

Another approach consists in replacing determinism by a weaker constraint that suffices in some particular context. In this spirit, Good-for-Games automata (GFG for short) were introduced in [9], as a way to solve the Church synthesis problem. This problem asks, given a specification L, typically given by an LTL formula, over an alphabet of inputs and outputs, whether there is a reactive system (transducer) whose behaviour is included in L. The classical solution computes a deterministic automaton for L, and solves a game defined on

this automaton. It turns out that replacing determinism by the weaker constraint of being GFG is sufficient in this context. Intuitively, GFG automata are non-deterministic automata where it is possible to build an accepting run in an online way, without knowledge of the future, provided the input word is in the language of the automaton. In [9], it is shown that GFG automata allow an incremental algorithm for the Church synthesis problem: we can build increasingly large games, with the possibility that the algorithm stops before the full determinisation is needed. One of the aims of this paper is to generalise this idea to determinisation of NFA, for use in any context and not only Church synthesis. We give an incremental determinisation construction, where the emphasis is on space-saving, and that allows in some cases to avoid building the full powerset construction.

The notion of width introduced in this paper generalises the GFG model, by allowing more than one run to be built in an online way. Intuitively, width quantifies how many states we have to keep track of simultaneously in order to build an accepting run in an online way. The maximal width of an automaton is its number of states. The width of an automaton corresponds to the number of steps performed by our incremental determinisation construction before stopping. In the worst case where the width is equal to the number of states of the automaton, we end up performing the full powerset construction (or its generalisations for infinite words). We study here the complexity of directly computing the width of a nondeterministic automaton, and we show that it is PSPACE-hard and in EXPTIME.

The properties of GFG automata and links with other models (tree automata, Markov Decision Processes) are studied in [2, 10, 11]. Colcombet introduced a generalisation of the concept of GFG called history-determinism [5], replacing determinism for automata with counters. It was conjectured by Colcombet [6] that GFG automata were essentially deterministic automata with additional useless transitions. It was shown in [11] that on the contrary there is in general an exponential state space blowup to translate GFG automata to deterministic ones. GFG automata retain several good properties of determinism, in particular they can be composed with trees and games, and easily checked for inclusion.

We give here the first algorithms allowing to build GFG automata from arbitrary non-deterministic automata on infinite words, allowing to potentially save exponential space compared to deterministic automata. Our incremental constructions look for small GFG automata, and aim at avoiding the worst-case complexities of determinisation constructions. Moreover, in the case of coBüchi automata, we show that the procedure is more efficient than its analog looking for a deterministic automaton, since checking for GFGness is polynomial [11], while we show here that the corresponding step for determinisation, that is checking whether a coBüchi automaton is Determinisable By Pruning (DBP) is NP-complete.

As a measure of non-determinism, width can be compared with ambiguity, where the idea is to limit the number of possible runs of the automaton. In this context unambiguous automata play a role analogous to GFG automata for width. Unambiguous automata are studied in [12], degrees of ambiguity are investigated in [18, 13, 14]. In the online long version of the paper, we give examples of automata with various width and ambiguity, showing that these two measures are essentially orthogonal.

We start by describing the width approach on finite words, and then move to infinite words, focusing mainly on the coBüchi acceptance condition. We end by briefly describing the picture for Büchi automata.

2 Definitions

We will use Σ to denote a finite alphabet. The empty word is denoted ε. If $i \leq j$, the set $\{i, i+1, i+2, \ldots, j\}$ is denoted $[i, j]$. If X is a set and $k \in \mathbb{N}$, we note $X^{\leq k}$ for $\bigcup_{i=0}^{k} X^i$. The complement of a set X is denoted $\overline{X}$. If $u \in \Sigma^*$ is a word and $L \subseteq \Sigma^*$ is a language, the left quotient of L by u is $u^{-1}L := \{v \in \Sigma^* \mid uv \in L\}$.

2.1 Automata

A non-deterministic automaton $\mathcal{A}$ is a tuple $(Q, \Sigma, q_0, \Delta, F)$ where Q is the set of states, Σ is a finite alphabet, $q_0 \in Q$ is the initial state, $\Delta : Q \times \Sigma \to 2^Q$ is the transition function, and $F \subseteq Q$ is the set of accepting states.

The transition function is naturally generalised to 2^Q by setting for any $(X, a) \in 2^Q \times \Sigma$ $\Delta(X, a)$ the set of a-successors of X, i.e. $\Delta(X, a) = \{q \in Q \mid \exists p \in X, q \in \Delta(p, a)\}$.

If for all $(p, a) \in Q \times \Sigma$ there is a unique $q \in Q$ such that $(p, a, q) \in \Delta$, we say that $\mathcal{A}$ is *deterministic*.

If $u = a_1 \ldots a_n$ is a finite word of Σ^*, a run of $\mathcal{A}$ on u is a sequence $q_0 q_1 \ldots q_n$ such that for all $i \in [1, n]$, we have $q_i \in \Delta(q_{i-1}, a_i)$. The run is said to be *accepting* if $q_n \in F$.

If $u = a_1 a_2 \ldots$ is an infinite word of Σ^ω, a run of $\mathcal{A}$ on u is a sequence $q_0 q_1 q_2 \ldots$ such that for all $i > 0$, we have $q_i \in \Delta(q_{i-1}, a_i)$. A run is said to be *Büchi accepting* if it contains infinitely many accepting states, and *coBüchi accepting* if it contains finitely many non-accepting states. Automata on infinite words will be called Büchi and coBüchi automata, to specify their acceptance condition.

We will note NFA (resp. DFA) for a non-deterministic (resp. deterministic) automaton on finite words, NBW (resp. DBW) for a non-deterministic (resp. deterministic) Büchi automaton, and NCW (resp. DCW) for a non-deterministic (resp. deterministic) coBüchi automaton.

We also mention the *parity condition* on infinite words: each state q has a rank $\mathrm{rk}(q) \in \mathbb{N}$, and an infinite run is accepting if the highest rank appearing infinitely often is even.

The language of an automaton $\mathcal{A}$, noted $L(\mathcal{A})$, is the set of words on which the automaton $\mathcal{A}$ has an accepting run. Two automata are said equivalent if they recognise the same language.

An automaton $\mathcal{A}$ is *determinisable by pruning* (DBP) if an equivalent deterministic automaton can be obtained from $\mathcal{A}$ by removing some transitions.

An automaton $\mathcal{A}$ is *Good-For-Games* (GFG) if there exists a function $\sigma\colon A^* \to Q$ (called *GFG strategy*) that resolves the non-determinism of $\mathcal{A}$ depending only on the prefix of the input word read so far: over every word $u = a_1 a_2 a_3 \ldots$ (finite or infinite depending on the type of automaton considered), the sequence of states $\sigma(\varepsilon)\sigma(a_1)\sigma(a_1 a_2)\sigma(a_1 a_2 a_3)\ldots$ is a run of $\mathcal{A}$ on u, and it is accepting whenever $u \in L(\mathcal{A})$. For instance every DBP automaton is GFG. See [2] for more introductory material and examples on GFG automata.

2.2 Games

A *game* $\mathcal{G} = (V_0, V_1, v_I, E, W_0)$ of infinite duration between two players 0 and 1 consists of: a finite set of *positions* V being a disjoint union of V_0 and V_1; an *initial position* $v_I \in V$; a set of *edges* $E \subseteq V \times V$; and a *winning condition* $W_0 \subseteq V^\omega$.

A *play* is an infinite sequence of positions $v_0 v_1 v_2 \cdots \in V^\omega$ such that $v_0 = v_I$ and for all $n \in \mathbb{N}$, $(v_n, v_{n+1}) \in E$. A play $\pi \in V^\omega$ is *winning* for Player 0 if it belongs to W_0. Otherwise π is *winning* for Player 1.

A *strategy* for Player 0 (resp. 1) is a function $\sigma_0 \colon V^* \times V_0 \to V$ (resp. $\sigma_1 \colon V^* \times V_1 \to V$), describing which edge should be played given the history of the play $u \in V^*$ and the current position $v \in V$. A strategy has to obey the edge relation, i.e. there has to be an edge in E from v to $\sigma_P(u, v)$. A play π is *consistent* with a strategy σ_P of a player P if for every n such that $\pi(n) \in V_P$ we have $\pi(n+1) = \sigma_P(v_0 \dots v_{n-1}, v_n)$.

A strategy for Player 0 (resp. Player 1) is *positional* if it does not use the history of the play, i.e. it is a function $V_0 \to V$ (resp. $V_1 \to V$).

We say that a strategy σ_P of a player P is *winning* if every play consistent with σ_P is winning for P. In this case, we say that P *wins* the game $\mathcal{G}$.

A game is *positionally determined* if exactly one of the players has a positional winning strategy in the game.

3 Finite words

3.1 Width of a NFA

Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ be a NFA, and $n = |Q|$ be the size of $\mathcal{A}$.

We want to define the *width* of a $\mathcal{A}$ as the minimum number of simultaneous states that need to be tracked in order to be able to deterministically build an accepting run in an online way.

In order to define this notion formally, we introduce a family of games $\mathcal{G}_w(\mathcal{A}, k)$, parameterized by an integer $k \in [1, n]$.

The game $\mathcal{G}_w(\mathcal{A}, k)$ is played on $Q^{\leq k}$, starts in $X_0 = \{q_0\}$, and the round i of the game from a position $X_i \in Q^{\leq k}$ is defined as follows:

- Player 1 chooses a letter $a_{i+1} \in \Sigma$.
- Player 0 moves to a subset $X_{i+1} \subseteq \Delta(X_i, a_{i+1})$ of size at most k.

A play is winning for Player 0 if for all $r \in \mathbb{N}$, whenever $a_1 a_2 \dots a_r \in L(\mathcal{A})$, X_r contains an accepting state.

▶ **Definition 1.** The width of a NFA $\mathcal{A}$, denoted $\mathrm{width}(\mathcal{A})$, is the least k such that Player 0 wins $\mathcal{G}_w(\mathcal{A}, k)$.

Intuitively, the width measures the "amount of non-determinism" in an automaton: it counts the number of simultaneous states we have to keep track of, in order to be sure to find an accepting run in an online way.

▶ **Fact 2.** *A NFA $\mathcal{A}$ is GFG if and only if* $\mathrm{width}(\mathcal{A}) = 1$.

3.2 Partial powerset construction

We give here a generalisation of the powerset construction, following the intuition of the width measure.

We define the k-subset construction of $\mathcal{A}$ to be the subset construction where the size of each set is bounded by k. Formally, it is the NFA $\mathcal{A}_k = (Q^{\leq k}, \Sigma, \{q_0\}, \Delta', F')$ where:

- $\Delta'(X, a) := \begin{cases} \{\Delta(X, a)\} & \text{if } |\Delta(X, a)| \leq k \\ \{X' \mid X' \subseteq \Delta(X, a), |X'| = k\} & \text{otherwise} \end{cases}$
- $F' := \{X \in Q^{\leq k} \mid X \cap F \neq \emptyset\}$

▶ **Lemma 3.** *$\mathcal{A}_k$ has less than* $\dfrac{n^k}{(k-1)!} + 1$ *states.*

Proof. The number of states of $\mathcal{A}_k$ is (at most) $|Q^{\leq k}| = \sum_{i=0}^{k} \binom{n}{i}$. Using the fact that $\binom{n}{i} \leq \frac{n^i}{i!}$, we can bound the number of states of $\mathcal{A}_k$ by $\sum_{i=0}^{k} \frac{n^i}{i!} \leq \sum_{i=0}^{k} \frac{n^k}{k!} \leq 1 + \sum_{i=1}^{k} \frac{n^k}{k!} = \frac{n^k}{(k-1)!} + 1$. ◀

The following lemma shows the link between width and the k-powerset construction.

▶ **Lemma 4.** *width$(\mathcal{A}) \leq k$ if and only if $\mathcal{A}_k$ is GFG.*

Proof. Winning strategies in $\mathcal{G}_w(\mathcal{A}, k)$ are in bijection with GFG strategies for $\mathcal{A}_k$. ◀

3.3 GFG automata on finite words

We recall here results on GFG automata on finite words.

We start with a Lemma characterizing GFG strategies. Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ be a NFA recognising a language L, and $\sigma : \Sigma^* \to Q$ be a potential GFG strategy. If $q \in Q$, we denote $L(q)$ the language accepted from q in $\mathcal{A}$, i.e. $L(q)$ is the language of $\mathcal{A}$ with q as initial state.

▶ **Lemma 5.** *σ is a GFG strategy if and only if for all $u \in \Sigma^*$, $L(\sigma(u)) = u^{-1}L$*

We now go to the main result of this section. This result has first been proved in [1], and then a more general version allowing lookahead was proved using a game-based approach in [15].

▶ **Theorem 6.** *[1, 15] A NFA $\mathcal{A}$ is GFG if and only if it is DBP. Moreover, it is in $O(n^2)$ to determine whether a NFA of size n is GFG, and to compute an equivalent DFA by removing transitions.*

3.4 Incremental determinisation procedure

We can now describe an incremental determinisation procedure, aiming at saving resources in the search of a deterministic automaton. In the process, we also compute the width of the input NFA.

The algorithm goes as follows:

Algorithm 1:

$k = 0$
Repeat
$k := k + 1$
Construct $\mathcal{A}_k$
Until $\mathcal{A}_k$ is GFG
Compute an equivalent DFA $\mathcal{D}$ from $\mathcal{A}_k$ by removing transitions
Return $\mathcal{D}, k$

The usual determinisation procedure uses the full powerset construction, i.e. assumes that we are in the case of maximal width. In a second step, the deterministic automaton can be minimized easily.

Our method here is to approach this construction "from below", and incrementally increase the width until we find the good one. In some cases, this allows to compute directly a smaller automaton, and avoiding using the full powerset construction of exponential state complexity.

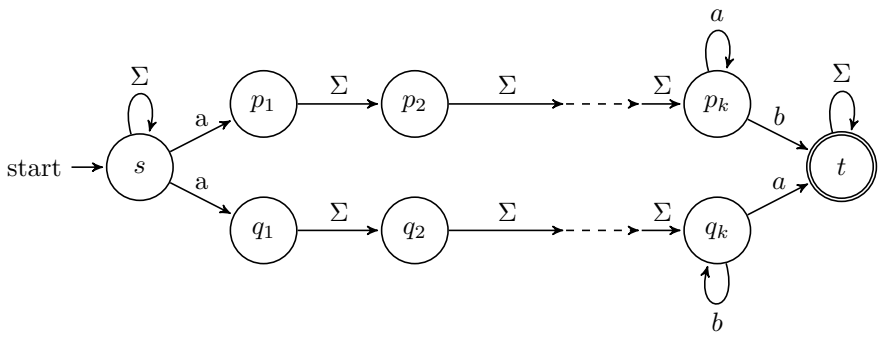


Figure 1 Example: 2-subset construction is enough.

For a NFA with n states and width k, the complexity of this algorithm is in $O\left(\frac{n^{2k}}{(k-1)!^2}\right)$, by Lemma 3 and Theorem 6.

▶ **Example 7.** Here the language recognised by this automaton is $L(\mathcal{A}) = \Sigma^* a \Sigma^{\geq k}$, and it has width 2. Therefore, our determinisation procedure uses time $O(n^4)$ and directly builds a DFA of size $O(n^2)$, while a classical determinisation via powerset construction would build an exponential-size DFA.

But in some other cases, the powerset construction is actually more efficient than the k-powerset construction, in terms of number of reachable states. It would therefore be interesting to be able to either run the two methods in parallel, or guess which one is more efficent based on the shape of the input NFA.

3.5 Complexity results on the width problem

In this section, we study the complexity of the *width problem*: given a NFA $\mathcal{A}$ and an integer k, is it true that $\text{width}(\mathcal{A}) \leq k$?

Being able to solve this problem efficiently would allow us to optimize the incremental determinisation algorithm, by aiming at the optimal k matching the width right away instead of trying the different width candidates incrementally.

▶ **Theorem 8.** *The width problem is PSPACE-hard.*

Proof. We prove this by reduction from the universality of NFA Problem (i.e. does an input NFA accept all words?) which is known to be PSPACE-Complete.

Let $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ be a NFA that we want to check for universality. Let $n = |Q|$.

Let a be a letter in Σ and $\#$ be a new letter not in Σ.

We build a NFA $\mathcal{B}$ over $\Sigma' = \Sigma \cup \{\#\}$ as the union of two NFAs $\mathcal{B}_1$ and $\mathcal{B}_2$ as shown in Figure 2.

Formally $\mathcal{B}_1 = (Q_1, \Sigma', q_0, \Delta_1, F_1)$, with $Q_1 = Q \cup \{q_\#\}$, $F_1 = \{q_\#\}$, and

$$\Delta_1(p,x) = \begin{cases} \Delta(p,x) & \text{if } p \in Q \text{ and } x \neq \# \\ \{q_\#\} & \text{if } (p,x) \in F \times \{\#\} \text{ or if } (p,x) = (q_\#, a) \\ \emptyset & \text{otherwise.} \end{cases}$$

Its language is $L(\mathcal{B}_1) = L(\mathcal{A})\#a^*$.

The NFA $\mathcal{B}_2 = (Q_2, \Sigma', p_I, \Delta_2, \{p_0\})$ has $n+2$ states as described on the picture, and recognises the language $L(\mathcal{B}_2) = \Sigma^* \# a^{\geq n}$.

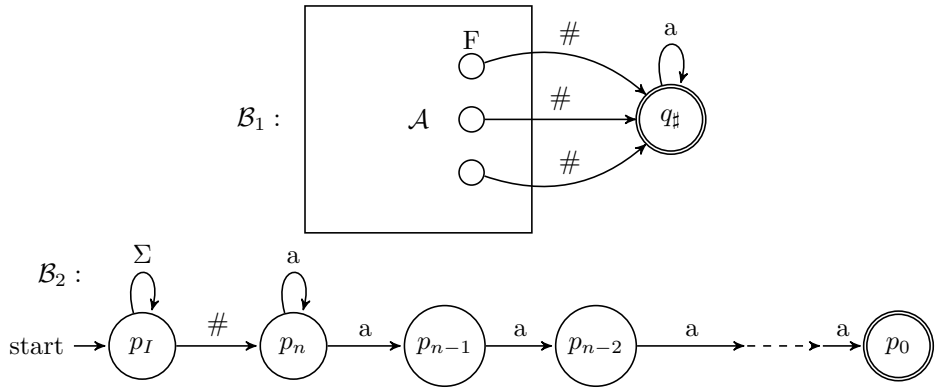


Figure 2 Automaton $\mathcal{B}$.

We define $\mathcal{B} = (Q_\mathcal{B}, \Sigma', \{q_0, p_I\}, \Delta_\mathcal{B}, F_\mathcal{B})$ as the union of $\mathcal{B}_1$ and $\mathcal{B}_2$. We allow here multiple initial states for simplicity, but it is straigthforward to adapt the construction in order to have a unique initial state.

The intuition here is that if $\mathcal{A}$ is universal, then $\mathcal{B}_2$ is useless in $\mathcal{B}$ as $L(\mathcal{B}) = L(\mathcal{B}_1) = \Sigma^*\#a^*$, and $\text{width}(\mathcal{B}) = \text{width}(\mathcal{A}) \leq n$. However, if $\mathcal{A}$ is not universal, then $\mathcal{B}$ is forced to use its $\mathcal{B}_2$ component, inducing a width at least $n+1$. This is formalized in the following lemma.

▶ **Lemma 9.** $\text{width}(\mathcal{B}) \leq n \iff L(\mathcal{A}) = \Sigma^*$.

Proof. ($\Rightarrow$) Suppose $\text{width}(\mathcal{B}) \leq n$ but $\exists u \in \Sigma^* \setminus L(\mathcal{A})$. Let $\mathcal{C}_n$ be the n-subset construction of $\mathcal{B}$. By Lemma 4, $\mathcal{C}_n$ is GFG, let $\sigma : \Sigma^* \to (Q_\mathcal{B})^{\leq n}$ be a GFG strategy of $\mathcal{C}_n$.

Consider the word $w = u\#a^n$. Note that $w \in L(\mathcal{B}_2) \setminus L(\mathcal{B}_1)$. Let $X \in (Q_\mathcal{B})^{\leq n}$ be the subset reached by σ on w, i.e. $X = \sigma(w)$. Notice that since $u \notin L(\mathcal{A})$, we have $X \subseteq (Q_2 \setminus \{p_I\})$, i.e. $X \subseteq \{p_0, p_1, \ldots, p_n\}$. Since $|X| \leq n$, there is $i \in [0, n]$ such that $p_i \notin X = \sigma(w)$. This means that $p_0 \notin \sigma(wa^i)$, hence $\sigma(wa^i)$ is not accepting. But this word is in $L(\mathcal{B})$ (as it is in $L(\mathcal{B}_2)$), this contradicts the fact that σ is a GFG strategy. Therefore it must be the case that $L(\mathcal{A}) = \Sigma^*$.

($\Leftarrow$) We now assume that $L(\mathcal{A}) = \Sigma^*$. A GFG strategy in $\mathcal{C}_n$ is given by following the powerset construction in $\mathcal{B}_1$, and ignoring $\mathcal{B}_2$. This shows that $\text{width}(\mathcal{B}) \leq n$. ◀

This constitutes a polynomial reduction from universality to the width problem, so the width problem is PSPACE-hard. Actually, we even showed that the particular case of checking n-width of an automaton of size $2n+3$ is PSPACE-hard. ◀

▶ **Theorem 10.** *The width problem is in EXPTIME.*

Proof. To show the EXPTIME upper bound, it suffices to build the game $\mathcal{G}_w(\mathcal{A}, k)$ of exponential size. Solving such a game is polynomial in the size of the game, so this algorithm runs in exponential time. Also note that the algorithm given in section 3.4 computes the width of a NFA in EXPTIME. ◀

We currently do not know if the width problem is complete for PSPACE or EXPTIME, and we leave this problem open.

▶ Remark. Although the present section deals with finite words, all results are immediately transferable to safety and reachability automata on infinite words. These automata have special acceptance conditions, which are particular cases of both Büchi and coBüchi conditions. Any infinite run is accepting in a safety automaton, and a run is accepting in a reachability automaton if it contains an accepting state. These dual acceptance conditions are of particular interest in verification, as they describe very natural properties.

4 CoBüchi Automata

We now turn to the case of coBüchi automata, and their determinisation problem. Here, since GFG and DBP are no longer equivalent [2, 11], we will also be interested in building GFG automata. As we will see, coBüchi automata are particularly well-suited for this approach for several reasons.

First of all, we recall that NCW and DCW have same expressive power, i.e. the determinisation of coBüchi automata does not need to introduce more complex acceptance conditions.

4.1 Width of ω-automata

We define here the width of automata on infinite words in a general way, as the definition is independent of the accepting condition.

Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, \alpha)$ be an automaton on infinite words with acceptance condition α, and $n = |Q|$ be the size of $\mathcal{A}$.

As before, we want to define the *width* of a $\mathcal{A}$ as the minimum number of states that need to be tracked in order to deterministically build an accepting run in an online way.

We will use the same family of games $\mathcal{G}_w(\mathcal{A}, k)$ as in Section 3.1, they will only differ in the winning condition.

The game $\mathcal{G}_w(\mathcal{A}, k)$ is played on $Q^{\leq k}$, starts in $X_0 = \{q_0\}$, and the round i of the game from a position $X_i \in Q^{\leq k}$ is defined as follows:

- Player 1 chooses a letter $a_{i+1} \in \Sigma$.
- Player 0 moves to a subset $X_{i+1} \subseteq \Delta(X_i, a_{i+1})$ of size at most k.

An infinite play is winning for Player 0 if whenever $a_1 a_2 \cdots \in L(\mathcal{A})$, the sequence $X_0 X_1 X_2 \ldots$ contains an accepting run. That is to say there is a valid accepting run $q_0 q_1 q_2 \ldots$ of $\mathcal{A}$ on $a_1 a_2 \ldots$ such that for all $i \in \mathbb{N}$, $q_i \in X_i$.

▶ **Definition 11.** The width of $\mathcal{A}$, denoted $\text{width}(\mathcal{A})$, is the least k such that Player 0 wins $\mathcal{G}_w(\mathcal{A}, k)$.

As before, an automaton $\mathcal{A}$ is GFG if and only if $\text{width}(\mathcal{A}) = 1$.

4.2 GFG coBüchi automata

We recall here some results from [11] on GFG coBüchi automata.

The first result is the exponential succinctness of coBüchi GFG automata compared to deterministic ones.

▶ **Theorem 12** ([11]). *There is a family of languages $(L_n)_{n\in\mathbb{N}}$ such that for all n, L_n is accepted by a coBüchi GFG automaton of size n, but any deterministic parity automaton for L_n must have size in $\Omega\left(\frac{2^n}{n}\right)$.*

Despite this apparent complexity of GFG NCW, the next theorem shows that they can be recognised efficiently.

▶ **Theorem 13** ([11]). *Given a NCW $\mathcal{A}$, it is in PTIME to decide whether $\mathcal{A}$ is GFG.*

The conjunction of these results make the coBüchi class particularly interesting in our setting: the succinctness allows us to potentially save a lot of space compared to classical determinisation, and Theorem 13 can be used to stop the incremental construction. This is in the context where we aim at building a GFG automaton, for instance in a context where we want to test for inclusion, or compose it with a game.

We examine later the case where GFG automata are not enough and we are aiming at building a DCW instead.

4.3 Partial breakpoint construction

We generalize here the breakpoint construction from [16], in the same spirit as Section 3.2.

For a parameter k, we want the k-breakpoint construction to be able to keep track of at most k states simultaneously.

Given a NCW $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$, we define the k-breakpoint construction of $\mathcal{A}$ as the NCW $\mathcal{A}_k = (Q', \Sigma, \Delta', (\{q_0\}, \{q_0\}), F')$, with

$Q' = \{(X, Y) | X, Y \in Q^{\leq k} \text{ and } Y \subseteq X\}$,

$$\Delta'((X,Y),a) := \begin{cases} \{(\Delta(X,a), \Delta(X,a))\} & \text{if } Y = \emptyset \text{ and } |\Delta(X,a)| \leq k \\ \{(X', X') |\ X' \subseteq \Delta(X,a), |X'| = k\} & \text{if } Y = \emptyset \text{ and } |\Delta(X,a)| > k \\ \{(\Delta(X,a), \Delta(Y,a) \cap F)\} & \text{if } Y \neq \emptyset \text{ and } |\Delta(X,a)| \leq k \\ \{(X', X' \cap (\Delta(Y,a) \cap F)) \mid X' \subseteq \Delta(X,a), |X'| = k\} & \text{otherwise} \end{cases}$$

$F' := \{(X, Y) \in Q' \mid Y \neq \emptyset\}$

That is, a run is accepting in $\mathcal{A}_k$ if it visits the states of the form $(X, \emptyset)$ finitely many times.

▶ **Lemma 14.** *The number of states of $\mathcal{A}_k$ is at most $\sum_{i=0}^{k} \binom{n}{i} 2^i$, which is in $O(\frac{(2n)^k}{k!})$.*

Proof. A state of $\mathcal{A}_k$ is of the form (X, Y) with $|X| \leq k$ and $Y \subseteq X$. Therefore, there are at most $\sum_{i=0}^{k} \binom{n}{i} 2^i$ such states. Since $\binom{n}{i} \leq \frac{n^i}{i!}$, we can bound the number of states by $\sum_{i=0}^{k} \frac{n^k}{k!} 2^i \leq \frac{n^k}{k!} 2^{k+1} = O(\frac{(2n)^k}{k!})$ ◀

▶ **Lemma 15.** *$L(A) = L(\mathcal{A}_k)$, and $\text{width}(\mathcal{A}) \leq k \iff \mathcal{A}_k$ is GFG.*

Proof. This amounts to verifying that the automaton $\mathcal{A}_k$ faithfully simulates the winning condition of $\mathcal{G}_w(\mathcal{A}, k)$. The proof naturally follows from the correctness proof of the breakpoint construction.

◀

4.4 Incremental construction of GFG NCW

Supppose we are given a NCW $\mathcal{A}$, and we want to build an equivalent GFG automaton.

We can do the same as in Section 3.4: incrementally increase k and test for GFGness of $\mathcal{A}_k$, which is in PTIME by Theorem 13. However in the coBüchi setting, the GFG automaton is not necessarily DBP, and can actually be more succinct than any deterministic automaton for the language (Theorem 12).

If we are in a context where we are satisfied with a GFG automaton, such as synthesis or inclusion testing, this procedure can provide us one much more efficiently than determinisation.

Indeed, the example from [11] showing that GFG NCW are exponentially succinct compared to deterministic automata can be easily generalized to any width. For instance if our procedure is applied to the product of this automaton from [11] with the one from Example 7, our construction will stop at the second step and generate a GFG automaton of quadratic size. This shows that the incremental construction for finding an equivalent GFG NCW can be very efficient compared to determinisation.

Directly computing the width of a NCW is PSPACE-hard and in EXPTIME, by the same arguments as in Section 3.1.

4.5 Aiming for determinism

In cases where a GFG automaton is not enough, and we want instead to build a DCW, we can test for DBPness instead of GFGness in the incremental algorithm. If we find the automaton is DBP, we can remove the useless transitions, and obtain an equivalent DCW.

Notice that the number of steps in this procedure corresponds to an alternative notion of width that can be called *det-width*. The det-width of an automaton $\mathcal{A}$ is the least k such that Player 0 has a positional winning strategy in $\mathcal{G}_w(\mathcal{A}, k)$. Det-width always matches width on finite words by Theorem 6, but the notions diverge on infinite words.

This section studies the complexity of checking DBPness for NCW. The next theorem shows that surprisingly, DBPness is harder to check than GFGness on NCW.

▶ **Theorem 16.** *Given a NCW $\mathcal{A}$, it is NP-complete to check whether it is DBP.*

We first show the hardness with the following lemma.

▶ **Lemma 17.** *Checking whether a NCW is DBP is NP-hard.*

Proof. We prove this by reduction from the Hamiltonian Cycle problem on a directed graph, which is known to be NP-complete.

Recall that a Hamiltonian cycle is a cycle using each vertex of the graph exactly once.

Suppose, we have a directed graph $G = ([1, n], E)$ and we want to check whether it contains a Hamiltonian cycle. W.l.o.g. we can assume that the graph is strongly connected, otherwise the answer is trivially no.

We construct a NCW $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$, where F is the set of accepting states, such that $\mathcal{A}$ is DBP if and only if G has a Hamiltonian cycle. The components of $\mathcal{A}$ are defined as follows: $Q := \bigcup_{i\in[1,n]}\{p_i, q_i, r_i\}$, $\Sigma := \{a_1, a_2, \cdots, a_n, \#\}$, $q_0 := p_1$, $F := \bigcup_{i\in[1,n]}\{p_i, q_i\}$, and finally Δ contains the following transitions, for all $i \in [1, n]$:

$$p_i \xrightarrow{a_i} q_i, \quad p_i \xrightarrow{a_j} r_i \text{ for all } j \neq i, \quad q_i \xrightarrow{\#} p_i, \quad \text{and } r_i \xrightarrow{\#} p_k \text{ if } (i, k) \in E \ .$$

The only non-determinism in $\mathcal{A}$ occurs at the r_i states when reading #: we then have a choice between all the p_k where $(i, k) \in E$.

We give an example for G in figure 3, where solid lines show the Hamiltonian cycle, and the construction of $\mathcal{A}$ from G in figure 4, where solid lines show a determinisation by pruning witnessing this Hamiltonian cycle.

For each $i \in [1, n]$, we can think of the set of states $\{p_i, q_i, r_i\}$ as a cloud in $\mathcal{A}$ representing the vertex i of the graph G.

Let $\Sigma' := \Sigma \setminus \{\#\}$, and $L = \bigcup_{i=1}^{n} (\Sigma'\#)^*(a_i\#)^\omega$. First note that, provided G is strongly connected, we have $L(\mathcal{A}) = L$. Indeed, for a run to be accepting by $\mathcal{A}$, it has to visit r_i

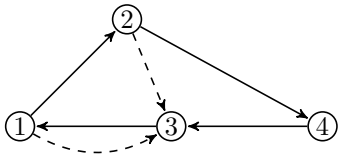


Figure 3 An instance of G.

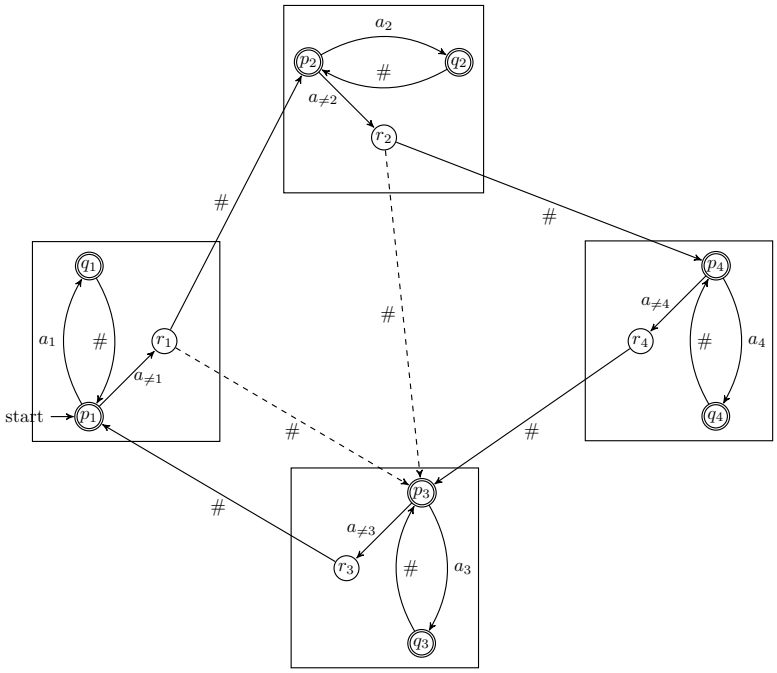


Figure 4 Construction of NCW $\mathcal{A}$ from G of figure 3.

finitely many times for all i, i.e. after some point it has to loop between p_i and q_i for some fixed i, so the input word must be in L. This shows $L(\mathcal{A}) \subseteq L$. On the other hand, consider a word $w \in L$ of the form $u(a_i\#)^\omega$ with $u \in (\Sigma'\#)^*$. Then $\mathcal{A}$ will have a run on u reaching some cloud j, and since the graph is strongly connected, the run can be extended to the cloud i reading a word of $(a_i\#)^*$. From there, the automaton will read $(a_i\#)^\omega$ while looping between p_i and q_i. We can build an accepting run of $\mathcal{A}$ on any word $w \in L$, so $L \subseteq L(\mathcal{A})$.

Now we shall prove that $\mathcal{A}$ is DBP if and only if G has a Hamiltonian cycle.

($\Rightarrow$) Suppose $\mathcal{A}$ is DBP, and let $\mathcal{D}$ be an equivalent DCW obtained from $\mathcal{A}$ by removing transitions. Notice that this corresponds to choosing one out-edge for each vertex of G. This means it induces a set of disjoint cycles in G. We show that it actually is a unique Hamiltonian cycle. Indeed, assume that some vertex of i is not reachable from 1 in G. Equivalently, it means that some cloud i is not reachable from p_1 in $\mathcal{D}$. This implies that $(a_i\#)^\omega \notin L(\mathcal{D})$, which contradicts $L(\mathcal{D}) = L(\mathcal{A}) = L$. Therefore, $\mathcal{D}$ is strongly connected, and describes a Hamiltonian cycle in G.

($\Leftarrow$) Conversely, if G has an Hamiltonian cycle π , we can build the automaton $\mathcal{D}$ accordingly, by setting for all $i \in [1, n]$, $\Delta_{\mathcal{D}}(r_i, \sharp) = \{p_j\}$ where j is the successor of i in π. Since $\mathcal{D}$ is strongly connected, it still recognises L, and since it is deterministic it is a witness that $\mathcal{A}$ is DBP.

This completes the proof of the fact that $\mathcal{A}$ is DBP if and only if G has a Hamiltonian cycle. Since this is a polynomial time reduction from Hamiltonian Cycle to DBPness of NCW, we showed that checking DBPness of a NCW is NP-hard.

Note that we used $n+1$ letters here, but it is straightfoward to re-encode this reduction using only two letters. Therefore, the problem is NP-hard even on a two-letters alphabet. It is trivially in PTIME on a one-letter alphabet, as there is a unique infinite word. ◀

The second part of Theorem 16 is given by the following lemma.

▶ **Lemma 18.** *Checking whether a NCW is DBP is in NP.*

Proof. Suppose a NCW $\mathcal{A}$ is given. We want to check whether it is DBP. We do this via the following NP algorithm.

- Nondeterministically prune transitions of $\mathcal{A}$ to get a deterministic automaton $\mathcal{D}$.
- Check whether $L(\mathcal{A}) \subseteq L(\mathcal{D})$. For that, we check if $L(\mathcal{A}) \cap \overline{L(\mathcal{D})} = \emptyset$

The second step of the algorithm can be done polynomially, since it amounts to finding an accepting lasso in $\mathcal{A} \times \overline{\mathcal{D}}$, where $\overline{\mathcal{D}}$ is a Büchi automaton obtained by dualizing the acceptance condition of $\mathcal{D}$. Finding such a lasso is actually in NL.

Therefore, the above algorithm is in NP, and its correctness follows from the fact that $L(\mathcal{D}) \subseteq L(\mathcal{A})$ is always true, as any run of $\mathcal{D}$ is in particular a run of $\mathcal{A}$.

◀

4.6 Towards Büchi automata

NBW corresponds to the general case of non-deterministic ω-automata, as they allow to recognise any regular language, and are easily computable from non-deterministic automata with stronger accepting conditions.

We will briefly describe the generalisation of previous constructions here, and explain what is the main open problem remaining to solve in order to obtain a satisfying generalisation. We take Safra's construction [17] as the canonical determinisation for Büchi automata. Safra's construction outputs a Rabin automaton.

The idea behind the previous partial determinisation construction can be naturally adapted to Safra: it suffices to restrict the image of the Safra tree labellings to sets of states of size at most k. The bottleneck of the incremental determinisation is then to test for GFGness (or DBPness) of Rabin automata. For DBPness, the same proof as Theorem 16 shows that it is NP-complete. However for GFGness, the complexity is widely open. The only known hardness result is the complexity of solving the games with same acceptance condition [11], known to be in QuasiP for parity [4] and NP-complete for Rabin [7]. In both cases, it is in P if the acceptance condition is fixed. On the other hand, the best known upper bound for GFGness is EXPTIME [11], even for fixed condition, say parity with 3 ranks. Finding an efficient algorithm for GFGness of Rabin (or Parity) automata would be of great interest for this incremental procedure, and would allow to efficiently build GFG automata from NBW.

References

1 Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 835–844, 2009.

2 Udi Boker, Denis Kuperberg, Orna Kupferman, and Michał Skrzypczak. Nondeterminism in the presence of a diverse or unknown future. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, pages 89–100, 2013.

3 Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, pages 457–468, 2013.

4 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 252–263, 2017.

5 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, languages and programming. Part II*, volume 5556 of *Lecture Notes in Comput. Sci.*, pages 139–150, Berlin, 2009. Springer.

6 Thomas Colcombet. Forms of determinism for automata (invited talk). In *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, pages 1–23, 2012.

7 E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs (extended abstract). In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 328–337, 1988.

8 Kousha Etessami. A hierarchy of polynomial-time computable simulations for automata. In *CONCUR 2002 - Concurrency Theory, 13th International Conference, Brno, Czech Republic, August 20-23, 2002, Proceedings*, pages 131–144, 2002.

9 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, pages 395–410, 2006.

10 Joachim Klein, David Müller, Christel Baier, and Sascha Klüppelholz. Are good-for-games automata good for probabilistic model checking? In *Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings*, pages 453–465, 2014.

11 Denis Kuperberg and Michał Skrzypczak. On determinisation of good-for-games automata. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 299–310, 2015.

12 H. Leung. Structurally unambiguous finite automata. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4094 LNCS:198–207, 2006. cited By 1.

13 Hing Leung. Separating exponentially ambiguous NFA from polynomially ambiguous NFA. In Kam-Wing Ng, Prabhakar Raghavan, N. V. Balasubramanian, and Francis Y. L. Chin, editors, *Algorithms and Computation, 4th International Symposium, ISAAC '93, Hong Kong, December 15-17, 1993, Proceedings*, volume 762 of *Lecture Notes in Computer Science*, pages 221–229. Springer, 1993.

14 HING LEUNG. Descriptional complexity of nfa of different ambiguity. *International Journal of Foundations of Computer Science*, 16(05):975–984, 2005.

15 Christof Löding and Stefan Repke. Decidability results on the existence of lookahead delegators for NFA. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*, pages 327–338, 2013.

16 Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω-words. *Theoret. Comput. Sci.*, 32(3):321–330, 1984.

17 Shmuel Safra. On the complexity of omega-automata. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 319–327, 1988.

18 Andreas Weber and Helmut Seidl. On the degree of ambiguity of finite automata. *Theoretical Computer Science*, 88(2):325–349, 1991.

Computing the Longest Common Prefix of a Context-free Language in Polynomial Time

Michael Luttenberger
Technische Universität München, Germany
luttenbe@in.tum.de

Raphaela Palenta
Technische Universität München, Germany
palenta@in.tum.de

Helmut Seidl
Technische Universität München, Germany
seidl@in.tum.de

Abstract

We present two structural results concerning the longest common prefixes of non-empty languages. First, we show that the longest common prefix of the language generated by a context-free grammar of size N equals the longest common prefix of the same grammar where the heights of the derivation trees are bounded by $4N$. Second, we show that each non-empty language L has a representative subset of at most three elements which behaves like L w.r.t. the longest common prefix as well as w.r.t. longest common prefixes of L after unions or concatenations with arbitrary other languages. From that, we conclude that the longest common prefix, and thus the longest common suffix, of a context-free language can be computed in polynomial time.

2012 ACM Subject Classification Mathematics of computing → Combinatorics on words, Theory of computation → Algebraic language theory, Theory of computation → Grammars and context-free languages

Keywords and phrases Longest Common Prefix, Context-free Languages, Combinatorics on Words

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.48

Related Version An extended version of this article is available at `https://arxiv.org/abs/1702.06698`, [11].

1 Introduction

Let Σ denote an alphabet. On the set Σ^* of all words over Σ, the prefix relation provides us with a partial ordering $\sqsubseteq$ defined by $u \sqsubseteq v$ iff $uu' = v$ for some $u' \in \Sigma^*$. The *longest common prefix* (lcp for short) of a non-empty set $L \subseteq \Sigma^*$ then is given by the greatest lower bound $\bigsqcap L$ of L w.r.t. this ordering. For two words $u, v \in \Sigma^*$, we also denote this greatest lower bound as $u \sqcap v$. Our goal is to compute the lcp when the language L is context-free, i.e., generated by a context-free grammar (CFG) — we therefore assume wlog. that Σ contains at least two letters.

The computation of the lcp (sometimes also *maximum common prefix*) is well studied for finite languages, in particular in the setting of string matching based on suffix arrays (e.g., [6]) where the string is given explicitly. Very often, strings can be efficiently compressed using *straight-line programs (SLPs)* — essentially CFGs which produce exactly one word. Interestingly, many of the standard string operations can still be done efficiently also on

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 48; pp. 48:1–48:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


SLP-compressed strings (see, e.g., [10]). As the union of SLPs is a (acyclic) CFG, the question of computing the lcp of a context-free language naturally arises. CFGs also represent a popular formalism to specify sets of well-formed words. Assume that we are given a CFG for the legal outputs of a program. This CFG might be derived from the specification as well as from an abstract interpretation of the program. Then the lcp of this language represents a prefix which can be output already, before the program actually has been run. This kind of information is crucial for the construction of *normal forms*, e.g., of string producing processors such as linear tree-to-string transducers [1, 8]. For these devices, the normal forms have further interesting applications as they allow for simple algorithms to decide equivalence [2] and enable efficient learning [9].

Obviously, the lcp of the context-free language L is a prefix of the shortest word in L. Since the shortest word of a context-free language can be effectively computed, the lcp of L is also effectively computable. The shortest word generated from a context-free grammar G, however, may be of length exponential in the size of G. Therefore, it is an intriguing question whether or not the lcp can be efficiently computed. Here, we show that the longest common prefix can in fact be computed in polynomial time. As the words the algorithm computes with may be of exponential length, we have to resort to *compressed* representations of long words by means of SLPs [12]. We will rely on algorithms for basic computational problems for SLPs as presented, e.g., in [10].

Our method of computing $\bigsqcap L$ is based on two structural results. First we show in Section 3 that it suffices to consider the finite sublanguage of L consisting of those words, for which there is a derivation tree of height at most $4N$ — with N the number of nonterminals for a CFG of L.[1] This implies that (1) in the proof of our main result we can replace the grammar by an *acyclic* context-free grammar, and (2) the actual fixpoint iteration to compute the lcp will converge within at most $4N$ iterations. Second we show in Section 4 that for every non-empty language L there is a subset $L' \subseteq L$ of at most three elements which is *equivalent* to L w.r.t. the lcp after arbitrary concatenations with other words. This means that for every word w, the language $L'w$ has the same lcp as Lw.

We illustrate both results by examples. For the first result, i.e. the restriction to derivation trees of bounded height, consider the language

$$L := \{a^2b(a^2b)^i a^2b(a^2ba)^i a^2ba^2ba^3 \mid i \in \mathbb{N}_0\}$$

generated by the context-free grammar consisting of the following rules over the alphabet $\Sigma = \{a, b, c\}$ and the six nonterminals $\{S, X, A_2, A_1, X_2, X_1\}$:

$$\begin{array}{llll} S \to X_2A_2bA_2bA_2a & A_2 \to aA_1 & A_1 \to a & X \to A_2b \\ & X_2 \to aX_1 & X_1 \to abX & X \to X_2A_2ba \end{array}$$

It is easy to check that here the lcp is already determined by repeating the derivation of X to $aabXaaba$ at most two times, which corresponds to the sublanguage consisting of all

[1] To simplify the presentation we assume that the CFG is proper, i.e. we will rule out production rules of the form $A \to B$ and $A \to \varepsilon$ (with A, B nonterminals and ε the empty word).

words which have a derivation tree of height at most 9.

$$\begin{aligned}
\bigsqcap L &= aabaabaabaabaa\,a && (i=0)\\
&\sqcap aabaabaabaabaa\,abaabaaa && (i=1)\\
&\sqcap aabaabaabaabaa\,baaabaaabaabaaa && (i=2)\\
&\sqcap aabaabaabaabaa\,baabaaabaaabaaabaabaaa && (i=3)\\
&\sqcap aabaabaabaabaa\,b\ldots && (i\geq 4)\\
&= aabaabaabaabaa
\end{aligned}$$

We remark that the bound of $4N$, i.e. 24 for this example, on the height resp. the number of iterations needed to converge is a crude overapproximation based on the pigeon-hole principle which does not take into account the structure of the grammar. The actual computation of the lcp may thus terminate much earlier, in particular when taking the dependency of nonterminals into account as done in Example 18.

In order to compute the lcp recursively, we call two languages $L_1, L_2 \subseteq \Sigma^*$ *equivalent w.r.t. the* lcp if for all words $w \in \Sigma^*$ we have that $\bigsqcap(L_1 w) = \bigsqcap(L_2 w)$. In Section 4 we show that every language L can be reduced to a sublanguage L' consisting of at most three words so that L and L' are equivalent w.r.t. the lcp . In fact, this result can be motivated by considering the special case of a language of the form $L = \{u, uv_1\}$ (with $u, v_1 \in \Sigma^*$) where we have $\bigsqcap(Lw) = u(w \sqcap v_1^\omega)$ for any $w \in \Sigma^*$ (see also Section 4). From this observation one immediately obtains that for finite languages $L' = \{uv_1, uv_2, \ldots, uv_k\}$ we have $\bigsqcap(L'w) = u(w \sqcap v_1^\omega \sqcap v_2^\omega \sqcap \ldots \sqcap v_k^\omega)$ and that one only needs to keep those two uv_i, uv_j for which $v_i^\omega \sqcap v_j^\omega$ is minimal. The result then extends to arbitrary languages. E.g., in case of the language $L = a(ba)^*$ we only need the sublanguage $\{a, aba\}$ (with $\varepsilon^\omega \sqcap (ba)^\omega := (ba)^\omega$) as the words a and aba suffice to characterize both $\bigsqcap L = a$ and the period ba that generates all suffices. For comparison, in case of $L = abab + aba(ba)^*$ the lcp is aba, which can only be extended to at most $abab = aba(b^\omega \sqcap (ba)^\omega)$. We therefore need to remember $\{aba, abab, ababa\}$: the sublanguages $\{aba, abab\}$ resp. $\{aba, ababa\}$ preserve $\bigsqcap L = aba$ but can be extended by b^ω resp. $(ba)^\omega$; whereas $\{abab, ababa\}$ only captures the maximal extension of $\bigsqcap L$, but does not preserve $\bigsqcap L$ itself.

In order to compute the lcp of a given context-free language L we then (implicitly) unfold the given context-free grammar into an acyclic grammar, and compute for every nonterminal of the unfolded grammar an equivalent sublanguage of at most three words, each compressed by means of a SLP, instead of the actual language. From this finite representation of L we then can easily obtain its lcp. Altogether, we arrive at a polynomial time algorithm.

Missing proofs can be found in the extended version of this article available on arxiv [11].

2 Preliminaries

Σ denotes a (finite) alphabet. We assume that Σ contains at least two letters as any context-free language over a unary alphabet is regular. Σ^* is the set of all finite words over Σ with ε the empty word, Σ^ω the set of all (countably) infinite words over Σ. We use (ω-)rational expressions to denote words and languages, e.g. $w^* = \varepsilon + w + ww + \ldots = \sum_{i\in\mathbb{N}_0} w^i$ and $w^\omega = wwwwwwwwww\ldots$.

By $\mathsf{C}_\Sigma = \{(u, v) \in \Sigma^* \times \Sigma^*\}$ we denote the set of all pairs of finite words over Σ. We define a multiplication on C_Σ by $(x, \bar{x})(y, \bar{y}) := (xy, \bar{y}\bar{x})$. For $(x, \bar{x}) \in \mathsf{C}_\Sigma$ and $w \in \Sigma^*$ set $(x, \bar{x})w = xw\bar{x}$. As in the case of words, we set $(x, \bar{x})^0 := (\varepsilon, \varepsilon)$, $(x, \bar{x})^{k+1} := (x, \bar{x})(x, \bar{x})^k$ and $(x, \bar{x})^* := \sum_{k\geq 0}(x, \bar{x})^k$ for all $x, \bar{x} \in \Sigma^*$ and $k \in \mathbb{N}_0$.

Note that we slightly deviate from standard notation when it comes to the prefix order (i.e. $u < w$) and the common prefix (i.e. $u \wedge v$) of two words in order to avoid the clash with

the notation for conjunction ($\wedge$): For $u, v \in \Sigma^*$ we write $u \sqsubseteq v$ ($u \sqsubset v$) to denote that u is a (strict) prefix of v, i.e. $v = uw$ for some $w \in \Sigma^*$ ($w \in \Sigma^+$). For $L \subseteq \Sigma^*$ (with $L \neq \emptyset$) its *longest common prefix* (lcp) $\bigsqcap L$ is given by the greatest lower bound of L w.r.t. this ordering. We simply write $u \sqcap v$ for $\bigsqcap\{u, v\}$. Note that for any word $w \in L$ there is at least one word $\alpha \in L$ s.t. $\bigsqcap L = w \sqcap \alpha$; we call any such α a *witness (w.r.t. w)*. Note that $\sqcap$ is commutative and associative; concatenation distributes from the left over the lcp (i.e. $u(v \sqcap w) = uv \sqcap uw$); and the lcp is monotonically decreasing on the union of languages, i.e. $\bigsqcap(L \cup L') = (\bigsqcap L) \sqcap (\bigsqcap L')$. The lcp of infinite words is defined analogously.

A word $p \in \Sigma^*$ is called a power of a word q if $p \in q^*$; then q is called a root of p; if $p \neq \varepsilon$ is its own shortest root, p it is called *primitive*. Two words u, v are *conjugates* if the is a factorization $u = pq$ and $v = qp$. We recall two well-known results:

▶ **Lemma 1** (Commutative Words, [3]). *Let $u, v \in \Sigma^*$ be two words. If $uv = vu$, then $u, v \in p^*$ for some primitive $p \in \Sigma^*$.*

▶ **Lemma 2** (Periodicity Lemma of Fine and Wilf, [5]). *Let $u, v \in \Sigma^+$ be two non-empty words. If $|u^\omega \sqcap v^\omega| \geq |u| + |v| - \gcd(|u|, |v|)$, then $uv = vu$.*

Combining these two lemmata yields the following result which is a useful tool in the proofs to follow (see also lemma 3.1 in [3] for a more general version of this result):

▶ **Corollary 3.** *Let $u, v \in \Sigma^*$ with $uv \neq vu$.*
Then $u^\omega \sqcap v^\omega = uv \sqcap vu$ with $|uv \sqcap vu| < |u| + |v| - \gcd(|u|, |v|)$.

Proof. Since the bound of the size of $|uv \sqcap vu|$ follows from Lemma 2 we only have to show that $uv \sqcap vu = u^\omega \sqcap v^\omega$. If $|u| = |v|$, then $uv \neq vu$ implies $u \neq v$ and $uv \sqcap vu = u \sqcap v = u^\omega \sqcap v^\omega$.

W.l.o.g. we assume that $|u| < |v|$. As $uv \neq vu$, we have $\varepsilon \neq u$. Let $v \sqcap u^\omega = u^k u' \sqsubset u^{k+1}$ with $v = u^k u' v'$ and $u = u'u''$. It follows that $uv \sqcap vu = uu^k u'v' \sqcap u^k u'v'u = u^k(uu'v' \sqcap u'v'u) = u^k u'(u''u'v' \sqcap v'u'u'')$.

If $v' \neq \varepsilon$, we have $u''u'v' \sqcap v'u'u'' = u'' \sqcap v' = \varepsilon$, and thus $uv \sqcap vu = u^k u' = v \sqcap u^\omega = v^\omega \sqcap u^\omega$.

So assume $v' = \varepsilon$, i.e. $v \sqsubset u^\omega$ with $k > 0$ as $|u| < |v|$. As $uv = u^k u'u''u' \neq u^k u'u'u'' = vu$, also $u'u'' \neq u''u'$. Hence $uv \sqcap vu = u^k u'(u''u' \sqcap u'u'') = u^{k+1}u \sqcap vv = u^\omega \sqcap v^\omega$, which concludes the proof. ◀

Here is a short example for the last corollary:

▶ **Example 4.** Let $u = aab$, $v = aaba = ua$. Then $uv \sqcap vu = aabaaba \sqcap aabaaab = aabaa = va$ and $u^\omega \sqcap v^\omega = aabaabaabu^\omega \sqcap aabaaabav^\omega = aabaa$ with $|aabaa| = |u| + |v| - \gcd(|u|, |v|) - 1$. I.e. the bound is sharp. Note that this example also shows, that even if $uv \neq vu$ and $\varepsilon \neq u \sqsubset v$, we still can have $v \sqsubset uv \sqcap vu$.

We briefly discuss properties of the lcp for very simple regular languages. These will be used several times in the proofs of Section 3 in order to bound the height of the derivation trees we need to consider:

▶ **Lemma 5.** *Let $y \neq \varepsilon$, then $w \sqcap yw = w \sqcap y^i w = \bigsqcap y^* w = w \sqcap y^\omega$ for all $i > 0$.*

Proof. Let $w \sqcap y^\omega = y^k y' \sqsubset y^{k+1}$ with $w = y^k y'w'$. Then for any $i > 0$ we have $w \sqcap y^i w = w \sqcap y^{k+i} y'w' = w \sqcap y^\omega$ where the last equality holds as $i > 0$ and $w \sqcap y^{k+1} = w \sqcap y^\omega \sqsubset y^{k+1}$. ◀

▶ **Lemma 6.** *If $w \not\sqsubseteq yw$, then $\bigsqcap y^* w = w \sqcap y^i w \sqsubset w$ for all $i > 0$.*

Proof. Since $w \not\sqsubseteq yw$, we have $w \neq \varepsilon$ and $y \neq \varepsilon$. By Lemma 5 we thus have $\bigsqcap y^* w = w \sqcap y^i w$ for any $i > 0$, in particular for $i = 1$. Define $w = y^k y'w'$ as in Lemma 5. As $w \not\sqsubseteq yw$, we have $w' \neq \varepsilon$ and thus $w \sqcap yw = y^k y' \sqsubset w$. ◀

We assume that the reader is familiar with context-free grammars (CFGs). We briefly introduce the notation we use for CFGs in the following. A context-free grammar G is given by a tuple $G = (\Sigma, V, P, S)$ where Σ is the alphabet of terminals, V is the set of nonterminals (also: variables), $P \subseteq V \times (V \cup \Sigma)^*$ is the set of production rules where a rule $p = (A, \gamma) \in P$ is also written as $A \to \gamma$, and S the axiom. The language generated by G is denoted by $L(G)$. G is *proper* if $A \to \varepsilon \notin P$ and $A \to B \notin P$ for all $A, B \in V$; G is in *Chomsky normal form (CNF)* if all rules are of the form $A \to a \in V \times \Sigma$ or $A \to BC \in V \to VV$. For every CFG G a proper CFG resp. a CFG in CNF G' can be constructed in time polynomial in the size of G such that $L(G) \setminus \{\varepsilon\} = L(G')$ [7]. As $\varepsilon \overset{?}{\in} L(G)$ is decidable in time polynomial in the size of G, and trivially $\bigsqcap L = \varepsilon$ if $\varepsilon \in L$, we will assume that $\varepsilon \notin L(G)$ and that G is proper from here on. For some proofs we assume in fact that G is in CNF but only in order to simplify notation.

3 LCP of a context-free language

Our main result in this section, Theorem 10, is that for every context-free language $L = L(G)$ generated by the given CFG G its lcp $\bigsqcap L$ is equal to the lcp of its *finite* sublanguage L' which contains only the words $w \in L$ which possess a derivation tree w.r.t. G whose height (considering only nonterminals) is at most four times the number of nonterminals of G. For the main result we require the following technical theorem (see the following example).

▶ **Theorem 7.** *Let* $L = (x, \bar{x})[(y_1, \bar{y}_1) + \ldots + (y_l, \bar{y}_l)]^* w$ *for* $(x, \bar{x}), (y_1, \bar{y}_1), \ldots, (y_l, \bar{y}_l) \in \mathsf{C}_\Sigma$ *and* $w \in \Sigma^*$. *Then:*

$$\bigsqcap L = \bigsqcap (x, \bar{x})[(y_1, \bar{y}_1)^{\leq 2} + \ldots + (y_k, \bar{y}_l)^{\leq 2}]w$$

Furthermore, if $\bigsqcap L = xw\bar{x} \sqcap xy^2 w \bar{y}^2 \bar{x} \sqsubset xw\bar{x} \sqcap xyw\bar{y}\bar{x}$ *for some* $(y, \bar{y}) \in \{(y_1, \bar{y}_1), \ldots, (y_l, \bar{y}_l)\}$, *then w.r.t. this* y *there exists some primitive* $q \in \Sigma^*$ *and some* $k > 0$ *such that*

$$yw = wq^k \wedge q\bar{y} \neq \bar{y}q \wedge \bigsqcap L = xw\bar{x} \sqcap xywq\bar{y}\bar{x} \wedge xwq^k(\bar{y} \sqcap q^\omega) \sqsubseteq \bigsqcap L \sqsubset xwq^{k+1}(\bar{y} \sqcap q^\omega)$$

The proof of the main theorem of this section, Theorem 10, crucially depends on the observation that in the case $\bigsqcap L \sqsubset xw\bar{x} \sqcap xyw\bar{y}\bar{x}$, all the words y_i are powers of the same primitive word p with $pw = wq$ and all that is needed to obtain a witness is one additional power of p resp. its conjugate q (with $pw = wq$) to which Theorem 7 refers to. We give an example in order to clarify the statement of Theorem 7 in the case of $l = 2 \wedge y_1 y_2 = y_2 y_1$ which is central to Theorem 10:

▶ **Example 8.** We write $(y, \bar{y})$ for $(y_1, \bar{y}_1)$ and $(z, \bar{z})$ for $(y_2, \bar{y}_2)$, respectively. Let $(x, \bar{x}) = (\varepsilon, ababaaa) = (\varepsilon, qqaaa)$, $(y, \bar{y}) = (ab, abaab) = (q, qaab)$, $(z, \bar{z}) = (ab, abaac) = (q, qaac)$, and $w = \varepsilon$ with $q = ab = y = z$. We then have:

$xw\bar{x}$	$=$	$ababaaa$
$xyw\bar{y}\bar{x}$	$=$	$ababaababababaaa$
$xzw\bar{z}\bar{x}$	$=$	$ababaacababaaa$
$xyyw\bar{y}\bar{y}\bar{y}\bar{x}$	$=$	$ababababaababaababababaaa$
$xyzw\bar{z}\bar{y}\bar{x}$	$=$	$ababababaacabaababababaaa$
$xzyw\bar{y}\bar{z}\bar{x}$	$=$	$ababababaababaacababaaa$
$xzzw\bar{z}\bar{z}\bar{x}$	$=$	$ababababaacabaacababaaa$
$x(y+z)^{\geq 3}\ldots$	$=$	$ababab\ldots$
$xywq\bar{y}\bar{x}$	$=$	$ababababababaaa$
$xzwq\bar{z}\bar{x}$	$=$	$ababababaacababaaa$
$\bigsqcap L$	$=$	$ababa$

So in this example, any word except for $xyw\bar{y}\bar{x}$ and $xzw\bar{z}\bar{x}$ is a witness for the lcp w.r.t. $xw\bar{x}$. W.r.t. the proof of Theorem 10 it is important that also in general we can pick a witness which either is derived using only $(y, \bar{y})$ or $(z, \bar{z})$ but not both, and that we need to use $(y, \bar{y})$ resp. $(z, \bar{z})$ at most twice in order to get one additional copy of the conjugate q of the primitive root of both y and z.

To give an impression of the proof of Theorem 7 we show the case $l = 1$. The complete proof of Theorem 7 can be found in the appendix of [11].

▶ **Lemma 9.** *Let $L = (x, \bar{x})(y, \bar{y})^* w$. Then: $\bigsqcap L = \bigsqcap (x, \bar{x})(y, \bar{y})^{\leq 2} w$.*
If $\bigsqcap L \sqsubset xw\bar{x} \sqcap xyw\bar{y}\bar{x}$, then there is some primitive q and some $k > 0$ s.t.

$$yw = wq^k \wedge q\bar{y} \neq \bar{y}q \wedge \bigsqcap L = xw\bar{x} \sqcap xywq\bar{y}\bar{x} \wedge xwq^k(\bar{y} \sqcap q^\omega) \sqsubseteq \bigsqcap L \sqsubset xwq^{k+1}(\bar{y} \sqcap q^\omega)$$

Proof. Recall that for any $z \in L$ there is some *witness* $z' \in L$ s.t. $\bigsqcap L = z \sqcap z'$. Our main goal is to show that w.r.t. $xw\bar{x}$ we find a witness within $\{xy^i w\bar{y}^i\bar{x} \mid i = 0, 1, 2\}$. What makes the proof technically more involved is that for Theorem 10 we need a stronger characterization of the case when $xyyw\bar{y}\bar{y}\bar{x}$ is the only witness in this set.

If $y = \varepsilon \vee \bar{y} = \varepsilon$, then L is actually regular and Lemma 5 already tells us that $xyw\bar{y}\bar{x}$ is a witness (w.r.t. $xw\bar{x}$). So wlog. $y \neq \varepsilon \neq \bar{y}$. If $w \not\sqsubseteq yw$, then $\bigsqcap y^* w = w \sqcap yw \sqsubset w$ by Lemma 6 and thus $\bigsqcap L = x(w \sqcap yw)$, i.e. $xyw\bar{y}\bar{x}$ is again a witness.

From now on we assume that $w \sqsubseteq yw$. Then there is some conjugate μ of y defined by $w\mu = yw$, and xw is a prefix of $\bigsqcap L$ as $xy^i w\bar{y}^i\bar{x} = xw\mu^i\bar{y}^i\bar{x}$. Wlog. we therefore assume $xw = \varepsilon$ from now on so that L becomes $\{y^i\bar{y}^i\bar{x} \mid i \in \mathbb{N}_0\}$.

Let q be the primitive root of y s.t. $y = q^k$ for a suitable $k > 0$ (as $y \neq \varepsilon$). By choosing $j > |\bar{x}| \,/\, |y|$ we obtain $\bigsqcap L \sqsubseteq \bar{x} \sqcap y^j\bar{y}^j\bar{x} = \bar{x} \sqcap q^{kj} \sqsubset q^\omega$, i.e. $\bigsqcap L \sqsubset q^\omega$. We therefore factorize $\bar{x}$ and $\bar{y}$ w.r.t. q^ω: Let $\bar{x} = q^n q'\bar{x}'$ with $\bar{x} \sqcap q^\omega = q^n q' \sqsubset q^{n+1}$; and let $\bar{y} = q^{k'}\hat{q}\bar{y}'$ with $\bar{y} \sqcap q^\omega = q^{k'}\hat{q} \sqsubset q^{k'+1}$. The words of L have thus the form $y^i\bar{y}^i\bar{x} = q^{ik}\left(q^{k'}\hat{q}\bar{y}'\right)^i q^n q'\bar{x}'$.

If q (resp. y) and $\bar{y}$ commute, then $\bar{y} = q^{k'}$ by Lemma 1 (as q is primitive) for some suitable $k' \in \mathbb{N}$. Then $L = (y\bar{y})^*\bar{x} = (q^{k+k'})^* q^n q'\bar{x}'$ with $\bigsqcap L = q^n q'$, and $y\bar{y}\bar{x}$ is again a witness w.r.t. $\bar{x}$. We thus also assume $q\bar{y} \neq \bar{y}q$ from here on.

If $q^n q' \sqsubseteq q^{k+k'}\hat{q}$, then $\bigsqcap L \sqsubseteq q^n q'$ and $qy\bar{y}\bar{x}$ is a witness w.r.t. $\bar{x}$: by choice of n we have $\bar{x} \sqcap q^\omega = \bar{x} \sqcap q^{n+1}$, by $q^n q' \sqsubseteq q^{k+k'}\hat{q}$ we also have $q^{n+1} \sqsubseteq q^{k+k'+1}$; from this we obtain $\bar{x} \sqcap qy\bar{y}\bar{x} = \bar{x} \sqcap q^{k+k'+1}\hat{q}\bar{x} = \bar{x} \sqcap q^{n+1} = q^n q'$. Thus, also $yy\bar{y}\bar{y}\bar{x}$ is a witness w.r.t $\bar{x}$. Assume now that $q^{k+k'}\hat{q} \sqsubset q^n q'$ and thus $q^{k+k'}\hat{q} \sqsubseteq \bigsqcap L$. If $\bigsqcap L = q^{k+k'}\hat{q}$, then $\bar{x} \sqcap y\bar{y}\bar{x} = q^{k+k'}\hat{q}$ has to hold, i.e. $y\bar{y}\bar{x}$ has to be a witness. Thus assume $q^{k+k'}\hat{q} \sqsubset \bigsqcap L$. If $\bar{y}' \neq \varepsilon$, then, as $q^{k+k'}\hat{q} \sqsubset q^n q'$, we have that $q^n q' \sqcap q^{k+k'}\hat{q}\bar{y}' = q^{k+k'}\hat{q}$ so that $y\bar{y}\bar{x}$ is again a witness. Hence assume $\bar{y}' = \varepsilon$ resp. $\bar{y} = q^{k'}\hat{q}$ for the remaining. As q and $\bar{y}$ do not commute, also q and $\hat{q}$ do not commute implying $q\hat{q} \sqsubset \hat{q}\, q \sqsubset q\hat{q}$. Thus

$$\begin{aligned} q^{k+k'}\hat{q} \sqsubset \bigsqcap L \sqsubseteq y\bar{y}\bar{x} \sqcap yy\bar{y}\bar{y}\bar{x} &= q^{k+k'}(\hat{q}q^n q'\bar{x}' \sqcap q^k\hat{q}\bar{y}\bar{x}) \\ &\overset{n \geq k > 0 \wedge \hat{q} \sqsubset q}{=} q^{k+k'}(\hat{q}q \sqcap q\hat{q}) \sqsubset q^{k+k'}q\hat{q} \end{aligned}$$

That is either $y\bar{y}\bar{x}$ or $yy\bar{y}\bar{y}\bar{x}$ has to be a witness w.r.t. $\bar{x}$ as $\bigsqcap L \sqsubset q^\omega$ and as we can extend $q^{k+k'}\hat{q}$ by at most $|q| - 1$ symbols, i.e. we need at most one additional copy of q which is again given by $yyw\bar{y}\bar{y}\bar{x}$ as $k > 0$. In particular, we have again that, if $yy\bar{y}\bar{y}\bar{x}$ is a witness, then so is $qy\bar{y}\bar{x}$. ◀

Using Theorem 7, we now can show that we only need to consider a finite sublanguage of L instead of L itself:

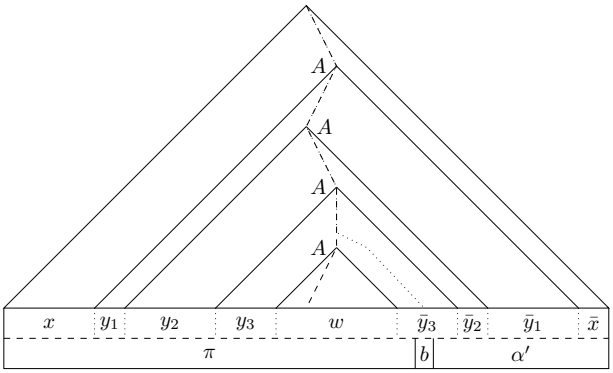


Figure 1 Factorization of a witness $\alpha = (x,\bar{x})(y_1,\bar{y}_1)(y_2,\bar{y}_2)(y_3,\bar{y}_3)w = \pi b\alpha'$ w.r.t. a nonterminal A occurring at least four times a long the dashed path in a derivation tree of α leading to a letter either within the lcp $\pi = \bigsqcap L$ or to the lcp-defining letter b (the leaf of the dotted path).

▶ **Theorem 10.** *Let $L = L(G)$ be given by a proper CFG $G = (\Sigma, V, P, S)$. Let $\hat{L} \subseteq L$ be the finite language of all words of L for which there is a derivation tree w.r.t. G of height[2] at most $4N$ with $N = |V|$. Then: $\bigsqcap L = \bigsqcap \hat{L}$.*

Proof. Let N be the number of nonterminals of G. Let $\sigma \in L$ be a shortest word, and $\alpha \in L$ a shortest word with $\bigsqcap L = \sigma \sqcap \alpha$. Set $\pi := \bigsqcap L$.

We claim that there is at least one such α (for any fixed σ) that has an derivation tree w.r.t. G of height less than $4N$.If $\sigma = \alpha$, we are done as σ has a derivation tree of height less than N. So assume $\sigma \neq \alpha$ s.t. $\sigma = \pi a\sigma'$ and $\alpha = \pi b\alpha'$ with $a \neq b$ and $a, b \in \Sigma$. Then fix any derivation tree t of α w.r.t. G.

In fact, we will show the stronger claim that any path from the root of t to any letter of πb has length at most $3N$ (i.e. all the paths leading to the separating letter b or a letter left of it, see Figure 1); note that any path that leads to a letter right of b (i.e. into α') has to enter a subtree of height less than N as soon as it leaves the path leading to b because of the minimality of α. Hence, if all the paths leading to b or a letter left of b have length less than $3N$, the longest path in the derivation tree must have length at most $4N$.

So assume for the sake of contradiction that there is a path leading to a letter within πb that has at least length $3N$ i.e. consists of at least $3N + 1$ nonterminals. Then there is one nonterminal A that occurs at least four times leading to a factorization

$$\alpha = (x,\bar{x})(y_1,\bar{y}_1)(y_2,\bar{y}_2)(y_3,\bar{y}_3)w$$

Note that $x\bar{x} \neq \varepsilon$, $y_i\bar{y}_i \neq \varepsilon$ $(i = 1, 2, 3)$, and $w \neq \varepsilon$ as G is proper. As this path ends at b or left of it, we have $xy_1y_2y_3 \sqsubseteq \pi$. With $(x,\bar{x})(y_i,\bar{y}_i)(y_j,\bar{y}_j)w \in L$ for any $i, j \in \{1, 2, 3\}$ we thus obtain that $xy_iy_j \sqsubseteq \pi$ and $xy_jy_i \sqsubseteq \pi$ and thus $y_iy_j = y_jy_i$ for all $i, j \in \{1, 2, 3\}$. So $y_i = p^{k_i}$ for the same primitive p using Lemma 1.

Let $L' = (x,\bar{x})[(y_1,\bar{y}_1) + (y_2,\bar{y}_2) + (y_3,\bar{y}_3)]^*w$ so that $\{xw\bar{x}, \alpha\} \subseteq L'$. By construction $L' \subseteq L$ and thus $\bigsqcap L \sqsubseteq \bigsqcap L' \sqsubseteq xw\bar{x} \sqcap \alpha$. As $xw\bar{x}$ is shorter than α, it cannot be a witness, so $\pi a \sqsubseteq xw\bar{x}$ and $\pi = xw\bar{x} \sqcap \alpha$. Hence

$$\bigsqcap L = \sigma \sqcap \alpha = \pi = xw\bar{x} \sqcap \alpha \sqsupseteq \bigsqcap L' \sqsupseteq \bigsqcap L \quad \text{i.e.} \quad \bigsqcap L = \bigsqcap L'$$

[2] We measure the height of a derivation tree only w.r.t. nonterminals along a path from the root to a leaf.

It therefore suffices to consider L' in the following; in particular, α has to be a witness w.r.t. $xw\bar{x}$ of minimal length, too. (From here on, *witness* will always be w.r.t. $xw\bar{x}$.) By virtue of Theorem 7 we have $\bigsqcap L' = \bigsqcap (x,\bar{x})[(y_1,\bar{y}_1)^{\leq 2} + (y_2,\bar{y}_2)^{\leq 2} + (y_3,\bar{y}_3)^{\leq 2}]w$. Note that $\bigsqcap L' \sqsubset xw\bar{x} \sqcap xy_iw\bar{y}_i\bar{x}$ for any $i = 1,2,3$ as $|xy_iw\bar{y}_i\bar{x}| < |\alpha|$ and thus $xy_iw\bar{y}_i\bar{x}$ cannot be a witness by minimality of α. So for some $I \in \{1,2,3\}$

$$\bigsqcap L' = xw\bar{x} \sqcap xy_Iy_Iw\bar{y}_I\bar{y}_I\bar{x} \sqsubseteq \alpha$$

i.e. $xy_Iy_Iw\bar{y}_I\bar{y}_I\bar{x}$ has to be also a witness. Set $(y,\bar{y}) := (y_I,\bar{y}_I)$ and $L'' = (x,\bar{x})(y,\bar{y})^*w$ so that $L'' \subseteq L' \subseteq L$ and $\bigsqcap L = \bigsqcap L' = \bigsqcap L''$ as

$$xw\bar{x} \sqcap xyyw\bar{y}\bar{y}\bar{x} = \bigsqcap L \sqsubseteq \bigsqcap L' \sqsubseteq \bigsqcap L'' \sqsubseteq xw\bar{x} \sqcap xyyw\bar{y}\bar{y}\bar{x} \sqsubset xyw\bar{y}\bar{x}$$

As $xyw\bar{y}\bar{x}$ is not a witness, Theorem 7 tells us that there is some q satisfying

$$yw = wq^k \wedge q\bar{y} \neq \bar{y}q \wedge \bigsqcap L = \bigsqcap L'' = xw\bar{x} \sqcap xywq\bar{y}\bar{x} \wedge xwq^k(\bar{y} \sqcap q^\omega) \sqsubseteq \bigsqcap L \sqsubset xwq^{k+1}(\bar{y} \sqcap q^\omega)$$

From this, we obtain: **1.** As we already know that $y_i = p^{k_i}$ (as they commute), it follows that p and q are conjugates with $pw = qw$ s.t. $y_iw = wq^{k_i}$. **2.** As $xwq^k \sqsubseteq \bigsqcap L \sqsubset xwq^\omega$, we find some $m \geq 0$ and $\dot{q} \sqsubset q$ s.t. $\pi = \bigsqcap L = xwq^kq^m\dot{q}$ and, thus, $\pi a = xwq^kq^m\dot{q}a \sqsubseteq xw\bar{x}$ and $\pi b = xwq^kq^m\dot{q}b \sqsubseteq xyyw\bar{y}\bar{y}\bar{x}$. (Here, b might change, yet it cannot become a as $xyyw\bar{y}\bar{y}\bar{x}$ is a witness.) Additionally, from $\pi = xw\bar{x} \sqcap xyyw\bar{y}\bar{y}\bar{x} \sqsubset xwq^{k+1}(\bar{y} \sqcap q^\omega)$ we obtain $\pi c \sqsubseteq xwq^{k+1}(\bar{y} \sqcap q^\omega)$, i.e. $q^m\dot{q}c \sqsubseteq q\bar{y} \sqcap q^\omega \sqsubset q^\omega$ and thus $\dot{q}c \sqsubseteq q$. Hence, any word with prefix $xwq^{k+1}(\bar{y} \sqcap q^\omega)$ is a witness.

If there was at least one $j \in \{1,2,3\} \setminus \{I\}$ with $k_j > 0$ s.t. $y_j = p^{k_j} \neq \varepsilon$, then $(x,\bar{x})(y_j,\bar{y}_j)(y,\bar{y})w$ would be a witness shorter than α as y_j would give us at least one copy of q:

$$\begin{array}{lcll} (x,\bar{x})(y_j,\bar{y}_j)(y,\bar{y})w & = & xy_jyw\bar{y}\bar{y}_j\bar{x} & \\ & \sqsupseteq & xwq^{k+k_j}\bar{y} & (\text{as } yw = wq^k \text{ and } y_jw = wq^{k_j}) \\ & \sqsupseteq & xwq^{k+k_j}(\bar{y} \sqcap q^\omega) & \\ & \sqsupseteq & xwq^{k+1}(\bar{y} \sqcap q^\omega) & (\text{as } k_j > 0 \text{ and } q^{k+1}(\bar{y} \sqcap q^\omega) \sqsubset q^\omega) \end{array}$$

So for all remaining $j \in \{1,2,3\} \setminus \{I\}$ we have $y_j = \varepsilon$ and thus $\bar{y}_j \neq \varepsilon$ as G is proper and thus $y_j\bar{y}_j \neq \varepsilon$. By Lemma 5 $\bigsqcap xw\bar{y}_j^*\bar{x} = xw\bar{x} \sqcap xw\bar{y}_j\bar{x}$, hence $\pi a \sqsubseteq xw\bar{y}_j^*\bar{x}$, i.e. $q^{k+m}\dot{q}a \sqsubseteq \bar{y}_j^\omega$. If $q^m\dot{q}b \sqsubseteq \bar{y}_j$ for some $j \in \{1,2,3\} \setminus \{I\}$ (recall $\dot{q}b \sqsubseteq q$), then as $a \neq b$

$$xw\bar{x} \sqcap (x,\bar{x})(y,\bar{y})(y_j,\bar{y}_j)w \overset{(\text{as } y_j = \varepsilon)}{=} xw(\bar{x} \sqcap q^k\bar{y}_j\bar{y}\bar{x}) = xw(q^{k+m}\dot{q}a \sqcap q^{k+m}\dot{q}b) = \pi$$

i.e. $xyy_jw\bar{y}_j\bar{y}\bar{x}$ would be a shorter witness than α. Hence $\bar{y}_j \sqsubseteq q^m\dot{q} \sqsubset q^{k+m}\dot{q}a$ for both $j \in \{1,2,3\} \setminus \{I\}$. Thus:

$$\left|q^\omega \sqcap \bar{y}_j^\omega\right| \geq \left|q^{k+m}\dot{q}\right| \geq |q| + |q^m\dot{q}| > |q| + |\bar{y}_j| - \gcd(|q|,|\bar{y}_j|)$$

By the periodicity lemma of Fine and Wilf (Lemma 2) this implies $\bar{y}_j = q^{k'_j}$ for some $k'_j > 0$ (as q primitive), and, subsequently as the final contradiction, that $xy_Iy_jw\bar{y}_j\bar{y}_I\bar{x}$ would be a shorter witness. ◀

4 Small Equivalent Subsets of Languages

In this section we formally introduce a notion of equivalence of languages w.r.t. longest common prefixes. The first main result of this section is that every non-empty language has

an equivalent subset consisting of at most three elements. In case of acyclic context-free languages, such a subset can be computed in polynomial time. In combination with Theorem 10, we can lift the restriction on acyclicity. This enables us to ultimately conclude that the longest common prefix of a context-free language can be computed in polynomial time.

▶ **Definition 11.** Two languages L, L' are *equivalent w.r.t the* lcp *(short:* $L \equiv L'$*)* iff $\bigsqcap(Lw) = \bigsqcap(L'w)$ for all words $w \in \Sigma^*$.

We observe that L is equivalent to L' w.r.t. the lcp also after union or concatenation from the left or right with arbitrary other languages. Formally, this amounts to the following properties:

▶ **Lemma 12.** *For all non-empty languages* $L, L', \hat{L}$ *with* $L \equiv L'$ *we have:*

1. $\bigsqcap(L\hat{L}) = \bigsqcap(L'\hat{L})$
2. $\bigsqcap(\hat{L}L) = \bigsqcap(\hat{L}L')$
3. $\bigsqcap(L \cup \hat{L}) = \bigsqcap(L' \cup \hat{L})$

Proof. The argument is as follows:

1. $\bigsqcap(L\hat{L}) = \bigsqcap_{w\in\hat{L}}(\bigsqcap(Lw)) = \bigsqcap_{w\in\hat{L}}(\bigsqcap(L'w)) = \bigsqcap(L'\hat{L})$;
2. $\bigsqcap(\hat{L}L) = \bigsqcap(\hat{L}(\bigsqcap L)) = \bigsqcap(\hat{L}(\bigsqcap L')) = \bigsqcap(\hat{L}L')$;
3. $\bigsqcap(L \cup \hat{L}) = \bigsqcap L \sqcap \bigsqcap \hat{L} = \bigsqcap L' \sqcap \bigsqcap \hat{L} = \bigsqcap(L' \cup \hat{L})$. ◀

The next lemma gives us an explicit formula for $\bigsqcap(Lw)$ for the special case of the two-element language $L = \{u, uv\}$.

▶ **Lemma 13.** *Assume that* $u, v \in \Sigma^*$ *with* $v \neq \epsilon$*. For all words* $w \in \Sigma^*$, $\bigsqcap(\{u, uv\}w) = u(w \sqcap v^\omega)$ *holds.*

Proof. $\bigsqcap(\{u, uv\}w) = uw \sqcap uvw$. If w and v are incomparable or w is a prefix of v, $w \sqcap vw = w \sqcap v = w \sqcap v^\omega$, and the claim follows. Thus, it remains to consider the case that $v \sqsubseteq w$. Then $w = v^i w'$ for some i so that v is no longer a prefix of w'. Then $\bigsqcap(\{u, uv\}w) = \bigsqcap(\{u, uv\}v^i w') = uv^i(w' \sqcap vw') = uv^i(w' \sqcap v^\omega) = u(w \sqcap v^\omega)$. ◀

The explicit formula from Lemma 13 can be used to identify small equivalent sublanguages.

▶ **Theorem 14.** *For every non-empty language* $L \subseteq \Sigma^*$ *there is a language* $L' \subseteq L$ *consisting of at most three words such that* $L \equiv L'$*.*

Proof. If L is a singleton language, we choose $L' = L$. So assume that L contains at least two words with lcp u. If the lcp u of L is not contained in L then we choose L' as consisting of the two minimal words w_1, w_2 so that $u = w_1 \sqcap w_2$. It remains to consider the case where the lcp u of L is contained in L. Then we have for each word $w \in \Sigma^*$,

$$\begin{aligned} \bigsqcap(Lw) &= \bigsqcap(\{uv \mid uv \in L\}w) \\ &= \bigsqcap\{\bigsqcap(\{u, uv\}w) \mid uv \in L, v \neq \epsilon\} \\ &= \bigsqcap\{u(w \sqcap v^\omega) \mid uv \in L, v \neq \epsilon\} \qquad \text{(Lemma 13)} \\ &= u(w \sqcap \bigsqcap\{v^\omega \mid uv \in L, v \neq \epsilon\}) \end{aligned} \tag{1}$$

If L is ultimately periodic, then all words in L are of the form uv_0^i for some $v_0 \in \Sigma^+$ and $i \geq 0$, and $(v_0^i)^\omega = v_0^\omega$. Thus, $\bigsqcap(Lw) = u(w \sqcap v^\omega)$ for any $uv \in L$ with $v \neq \epsilon$. Hence, $L \equiv L' = \{u, uv\}$ for any such v.

If L is not ultimately periodic, then we choose words $uv_1, uv_2 \in L$ so that the lcp of v_1^ω and v_2^ω has minimal length. Then

$$\begin{aligned} \bigsqcap(\{u, uv_1, uv_2\}w) &= u(w \sqcap v_1^\omega \sqcap v_2^\omega) \\ &= u(w \sqcap \bigsqcap\{v^\omega \mid uv \in L, v \neq \epsilon\}) \end{aligned}$$

by the minimality of $v_1^\omega \sqcap v_2\omega$. Therefore, $L \equiv L' = \{u, uv_1, uv_2\}$. ◀

Since for any non-empty words w_1, w_2 given by SLPs, an SLP for $w_1^\omega \sqcap w_2^\omega = w_1w_2 \sqcap w_2w_1$ (if $w_1 \neq w_2$) can be computed in polynomial time[3], we have:

▶ **Corollary 15.** *For every non-empty finite $L \subseteq \Sigma^*$ consisting of words each of which is represented by an SLP, a subset $L' \subseteq L$ consisting of at most three words can be calculated in polynomial time such that $L \equiv L'$.*

Proof. The proof distinguishes the same cases as in the proof of Theorem 14 and relies on polynomial algorithms on SLPs [10]. If L contains at most three words we are done. Since the words in L are given as SLPs, we can calculate (a SLP for) the lcp u of the words in L. Next, we determine whether u is in L. This can again be checked in polynomial time. If this is not the case, then we can select two words $w_1, w_2 \in L$ so that $u = w_1 \sqcap w_2$ giving us $L' = \{w_1, w_2\}$ in polynomial time. So, now assume that u is in L. Next, we check whether or not L is ultimately periodic, i.e., whether for any non-empty words v_1, v_2 with $uv_1, uv_2 \in L$, $v_1^\omega = v_2^\omega$. By Lemma 2 this is the case iff $v_1v_2 = v_2v_1$. The latter can be checked in polynomial time as concatenation and equality of SLPs can be calculated in polynomial time. If this is the case, then we obtain $L' = \{u, uv\}$ for some $uv \in L$ with $v \neq \epsilon$ in polynomial time.

It remains to consider the case where the lcp u is contained in L and L is not ultimately periodic. Then we need to determine words uv_1 and uv_2 in L with $v_1 \neq \epsilon \neq v_2$ such that $v_1^\omega \sqcap v_2^\omega$ has minimal length. Since $v_1^\omega \sqcap v_2^\omega = v_1v_2 \sqcap v_2v_1$ (see Corollary 3), such a pair can be computed in polynomial time as well. Therefore, $L' = \{u, uv_1, uv_2\}$ can be computed in polynomial time. ◀

The following lemma explains that equivalence of two non-empty languages of cardinalities at most 3 can be decided in polynomial time.

▶ **Lemma 16.** *Let $L_1, L_2 \subseteq \Sigma^*$ denote non-empty languages consisting of at most three words each, which are all given by SLPs. Then $L_1 \stackrel{?}{\equiv} L_2$ can be decided in polynomial time.*

Proof. If one of the two languages contains just a single word, then $L_1 \equiv L_2$ iff $L_1 = L_2$ — which can be decided in polynomial time. Otherwise, we first compute $\bigsqcap L_1$ and $\bigsqcap L_2$. If these differ, then by definition L_1 cannot be equivalent to L_2. Therefore assume now that $u = \bigsqcap L_1 = \bigsqcap L_2$ is the common lcp .

Obviously, L_i and $L_i \cup \{u\}$ are equivalent w.r.t. the lcp ($i = 1, 2$). Thus, for testing equality, we may add u to L_1 resp. L_2, if it is missing, and reduce L_1 resp. L_2 subsequently to languages of at most three words.

[3] Lohrey [10] gives an overview over the classical algorithms for SLPs. The fully compressed pattern matching problem for SLPs is in PTIME [10, Theorem 12], i.e. we can test whether one SLP is a factor of another SLP. Especially we can test whether one SLP is a prefix of another SLP. As we can build an SLP for any prefix of an SLP in polynomial time we can use a binary search to compute the lcp of two SLPs in polynomial time.

From Equation 1 follows that $L_1 \equiv L_2$ if $\bigsqcap\{v_1^\omega \mid uv_1 \in L_1, v_1 \neq \epsilon\} = \bigsqcap\{v_2^\omega \mid uv_2 \in L_2, v_2 \neq \epsilon\}$. This is the case if either $v_1^\omega = v_2^\omega$ for all $uv_1 \in L_1$ and $uv_2 \in L_2$ or for $uv_i, uv_i' \in L_i$, $v_i \neq \epsilon \neq v_i'$ with $w_i = v_i^\omega \sqcap v_i'^\omega$ is minimal for L_i $(i = 1, 2)$, $w_1 = w_2$ holds.

In the first case $v_1^\omega = v_2^\omega$ for all $uv_1 \in L_1$ and $uv_2 \in L_2$ can be checked in polynomial time according to the periodicity lemma of Fine and Wilf (cf. Corollary 3). In the second case w_1, w_2 can be computed and compared in polynomial time as all words are given as SLPs. Thus, we ultimately arrive at a polynomial time decision procedure. ◀

▶ Remark. Note that in light of the equivalence test, we can choose distinct letters $a, b \in \Sigma$, and equivalently replace the language $L_1 = \{uv_1, uv_2\}$ with $L_1' = \{ua, ub\}$ whenever $v_1 \neq \epsilon \neq v_2$ and $v_1 \sqcap v_2 = \epsilon$, and the language $L_2 = \{u, uv_1, uv_2\}$ by the language $L_2' = \{u, uwa, uwb\}$ whenever $w = v_1v_2 \sqcap v_2v_1 \neq v_1v_2$ holds. This reduced representation allows for an easier computation.

Now we have all pre-requisites to prove the main theorem of our paper.

▶ **Theorem 17.** *Assume that G is a proper context-free grammar with $L = L(G)$ non-empty. Then the longest common prefix of L can be calculated in polynomial time.*

Proof. Assume w.l.o.g. that G is a CFG in Chomsky normal form as this simplifies the notation. For the actual fixed-point iteration this is not required. Then we calculate $\bigsqcap L(G)$ as follows. We build (implicitly, see the following remark) an acyclic CFG $\hat{G}$ in polynomial time such that $L(\hat{G})$ consists of all words of $L(G)$ for which there is a derivation tree of height at most $4N$ where N is the number of nonterminals in G. To this end, we tag the variables with a counter that bounds the height of the derivation trees. In more detail, for every rewriting rule $A \to BC$ of G and every $i \in \{1, \ldots, 4N\}$ we add to $\hat{G}$ the rule $A^{(i)} \to B^{(i-1)}C^{(i-1)}$, and for every rule $A \to a$ of G and every $i \in \{0, 1, \ldots, 4N\}$ we add the rule $A^{(i)} \to a$ to $\hat{G}$. In a derivation tree w.r.t. $\hat{G}$ every path starting at some node labeled by $A^{(i)}$ has thus length at most i as i has strictly decreases when moving down to towards the leaves, hence, a node labeled by $A^{(i)}$ can only be the root of a (sub-)tree of height at most i. Further, every derivation tree of $\hat{G}$ becomes a derivation tree of G by simply replacing $A^{(i)}$ by A. As every rule of G is copied at most $4N + 1$ times with N the number of nonterminals of G, the size of $\hat{G}$ grows at most quadratically with the size of G. In particular, $\hat{G}$ is still proper and in CNF. For more details, see e.g. section 3 in [4].

By Theorem 10, we know that $\bigsqcap L(G) = \bigsqcap L(\hat{G})$. By construction, $\hat{G}$ is also in Chomsky normal form. For i from 0 to (at most) $4N$ (with N still the number of variables of the original grammar G – as $\hat{G}$ is acyclic we only need to compute $[A^{(i)}]$ once when proceeding bottom-up), we then compute in every iteration for every nonterminal $A^{(i)}$ (for the currently value of i) first the language

$$[A^{(i)}]' := \{a \in \Sigma^* \mid A^{(0)} \to a \in P\} \cup \bigcup_{A \to BC \in G} [B^{(i-1)}] \cdot [C^{(i-1)}]$$

By induction on i, we may assume that the languages $[B^{(i-1)}], [C^{(i-1)}]$ (a) have already been computed, (b) consist of at most three words, and (c) every word is given as an SLP. Note that the cardinality of every language $[A^{(i)}]'$ is polynomial in the size of G. By virtue of Corollary 15, we therefore can reduce $[A^{(i)}]'$ in polynomial time to a language $[A^{(i)}] \subseteq [A^{(i)}]'$ with $[A^{(i)}] \equiv [A^{(i)}]'$ and $\left|[A^{(i)}]\right| \leq 3$. By construction, we then have

$$[A^{(i)}] \quad \equiv \quad \{w \in \Sigma^* \mid A^{(i)} \Rightarrow^* w\}$$

Since $\hat{G}$ has polynomially many nonterminals only, the overall algorithm runs in polynomial time. ◀

▶ Remark. Note that we can drop the assumption that the grammars G and likewise $\hat{G}$ are in Chomsky normal form if the right-hand sides of all rules have bounded lengths. Then the cardinality of the languages $[A^{(i)}]'$ are still polynomial. Further, instead of spelling out the grammar $\hat{G}$ explicitly, we may perform a round robin fixpoint iteration where in every round we first compute

$$[A]' := \bigcup_{A \to w_1 B_1 w_2 B_2 \dots w_k B_k w_{k+1}} \{w_1\} \cdot [B_1] \cdot \{w_2\} \cdot [B_2] \cdots \{w_k\} \cdot [B_k] \cdot \{w_{k+1}\}$$

with initially $[A] := \{w \in \Sigma^* \mid A \to w \in G\}$, then updating $[A]$ so that $[A] \subseteq [A]'$ with $[A] \equiv [A]'$ and $|[A]| \leq 3$. Theorem 10 guarantees that the lcp is attained after at most $4N$ iterations. Using standard approaches like work lists, we only need to recompute $[A]$ if there is some rule $A \to \gamma B \delta$ in G and $[B]$ has changed since the last recomputation of $[A]$. As shown in Lemma 16 we can easily check if $[B] \not\equiv [B]'$ in every round and accordingly insert A into the work list.

We demonstrate this simplified version of the algorithm described in Theorem 17 by an example.

▶ **Example 18.** Consider the following grammar G with the following rules:

$$S \to Aababaac \mid ababaac \qquad A \to ab\,A\,abaab \mid ab\,A\,abaac \mid ababaab \mid ababaac$$

The round robin fixpoint iteration would proceed by iteratively evaluating the equations

$$\begin{aligned} [A]' &:= \{abwabaab, abwabaac, ababaab, ababaac \mid w \in [A]\} \\ [S]' &:= \{wababaac, ababaac \mid w \in [A]\} \end{aligned}$$

and recomputing the languages $[A]$ and $[S]$ so that $[A] \equiv [A]'$ and $[S] \equiv [S]'$ and both $[A]$ and $[S]$ consist of at most three words where we further reduce the words of $[A]$ and $[S]$ as described in the remark following Lemma 16. As $[A]$ does not depend on $[S]$, we can postpone the computation of $[S]$ until after $[A]$ has converged. In the first round, we have:

$$[A]' = \{ababaab, ababaac\}$$

and thus update $[A]$ to $[A] := \{(ab)^2aab, (ab)^2aac\}$. For the second round, we obtain

$$\begin{aligned} [A]' = ab\{(ab)^2aab, (ab)^2aac\}abaab \cup ab\{(ab)^2aab, (ab)^2aac\}abaac \cup \{(ab)^2aab, (ab)^2aac\} \\ \equiv \{(ab)^3a(ab)^2aab, (ab)^2aab\} \equiv \{(ab)^3, (ab)^2aa\} =: [A] \end{aligned}$$

which is already the fixpoint as an additional iteration would show. Therefore we obtain

$$\begin{aligned} [S]' &= \{(ab)^3, (ab)^2aa\}(ab)^2aac \cup \{(ab)^2aac\} \\ &\equiv \{(ab)^3(ab)^2aac, (ab)^2aac\} \equiv \{(ab)^3, (ab)^2aa\} =: [S] \end{aligned}$$

So $\bigsqcap L = (ab)^3 \sqcap (ab)^2aa = (ab)^2a$.

5 Conclusion

We have shown that the longest common prefix of a non-empty context-free language can be computed in polynomial time. This result was based on two structural results, namely, that it suffices to consider words with derivation trees of bounded height, and second that each non-empty language is equivalent to a sublanguage consisting of at most three elements. For the actual algorithm, we relied on succinct representations of long words by means of SLPs. It remains as an intriguing open question whether the presented method can be generalized to more expressive grammar formalisms.

References

1 Adrien Boiret. Normal form on linear tree-to-word transducers. In Adrian-Horia Dediu, Jan Janousek, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings*, volume 9618 of *Lecture Notes in Computer Science*, pages 439–451. Springer, 2016. doi:10.1007/978-3-319-30000-9_34.

2 Adrien Boiret and Raphaela Palenta. Deciding equivalence of linear tree-to-word transducers in polynomial time. In Srecko Brlek and Christophe Reutenauer, editors, *Developments in Language Theory - 20th International Conference, DLT 2016, Montréal, Canada, July 25-28, 2016, Proceedings*, volume 9840 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2016. doi:10.1007/978-3-662-53132-7_29.

3 Christian Choffrut and Juhani Karhumäki. *Combinatorics of Words*, pages 329–438. Springer Berlin Heidelberg, 1997. doi:10.1007/978-3-642-59136-5_6.

4 Javier Esparza and Michael Luttenberger. Solving fixed-point equations by derivation tree analysis. In Andrea Corradini, Bartek Klin, and Corina Cîrstea, editors, *Algebra and Coalgebra in Computer Science - 4th International Conference, CALCO 2011, Winchester, UK, August 30 - September 2, 2011. Proceedings*, volume 6859 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2011. doi:10.1007/978-3-642-22944-2_2.

5 N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. URL: http://www.jstor.org/stable/2034009.

6 Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Combinatorial Pattern Matching, 12th Annual Symposium, CPM 2001 Jerusalem, Israel, July 1-4, 2001 Proceedings*, pages 181–192, 2001.

7 Martin Lange and Hans Leiß. To CNF or not to cnf? an efficient yet presentable version of the CYK algorithm. *Informatica Didactica*, 8, 2009. URL: http://www.informatica-didactica.de/cmsmadesimple/index.php?page=LangeLeiss2009.

8 Grégoire Laurence, Aurélien Lemay, Joachim Niehren, Slawek Staworko, and Marc Tommasi. Normalization of sequential top-down tree-to-word transducers. In Adrian-Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications - 5th International Conference, LATA 2011, Tarragona, Spain, May 26-31, 2011. Proceedings*, volume 6638 of *Lecture Notes in Computer Science*, pages 354–365. Springer, 2011. doi:10.1007/978-3-642-21254-3_28.

9 Grégoire Laurence, Aurélien Lemay, Joachim Niehren, Slawek Staworko, and Marc Tommasi. Learning sequential tree-to-word transducers. In Adrian-Horia Dediu, Carlos Martín-Vide, José Luis Sierra-Rodríguez, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings*, volume 8370 of *Lecture Notes in Computer Science*, pages 490–502. Springer, 2014. doi:10.1007/978-3-319-04921-2_40.

10 Markus Lohrey. Algorithmics on slp-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012. doi:10.1515/gcc-2012-0016.

11 Michael Luttenberger, Raphaela Palenta, and Helmut Seidl. Computing the longest common prefix of a context-free language in polynomial time. *CoRR*, abs/1702.06698, 2017. arXiv:1702.06698.

12 Wojciech Plandowski. Testing equivalence of morphisms on context-free languages. In Jan van Leeuwen, editor, *Algorithms - ESA '94, Second Annual European Symposium, Utrecht, The Netherlands, September 26-28, 1994, Proceedings*, volume 855 of *Lecture Notes in Computer Science*, pages 460–470. Springer, 1994. doi:10.1007/BFb0049431.

Surjective H-Colouring over Reflexive Digraphs

Benoît Larose
LACIM, Université du Québec a Montréal, Canada

Barnaby Martin
Department of Computer Science, Durham University, U.K.

Daniël Paulusma
Department of Computer Science, Durham University, U.K.

Abstract

The Surjective H-Colouring problem is to test if a given graph allows a vertex-surjective homomorphism to a fixed graph H. The complexity of this problem has been well studied for undirected (partially) reflexive graphs. We introduce *endo-triviality*, the property of a structure that all of its endomorphisms that do not have range of size 1 are automorphisms, as a means to obtain complexity-theoretic classifications of Surjective H-Colouring in the case of reflexive *digraphs*. Chen [2014] proved, in the setting of constraint satisfaction problems, that Surjective H-Colouring is NP-complete if H has the property that all of its polymorphisms are essentially unary. We give the first concrete application of his result by showing that every endo-trivial reflexive digraph H has this property. We then use the concept of endo-triviality to prove, as our main result, a dichotomy for Surjective H-Colouring when H is a reflexive tournament: if H is transitive, then Surjective H-Colouring is in NL, otherwise it is NP-complete. By combining this result with some known and new results we obtain a complexity classification for Surjective H-Colouring when H is a partially reflexive digraph of size at most 3.

2012 ACM Subject Classification Mathematics of computing → Graph coloring, Theory of computation → Problems, reductions and completeness, Theory of computation → Graph algorithms analysis

Keywords and phrases Surjective H-Coloring, Computational Complexity, Algorithmic Graph Theory, Universal Algebra, Constraint Satisfaction

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.49

Funding BL was supported by NSERC and FRQNT. DP was supported by The Leverhulme Trust (RPG-2016-258).

Acknowledgements We thank Jan Bok for fruitful discussions as well as three anonymous reviewers for useful comments.

1 Introduction

The classical *homomorphism problem*, also known as H-Colouring, involves a fixed structure H, with input another structure G, of the same signature, invoking the question as to whether there is a function from the domain of G to the domain of H that is a homomorphism from G to H. The H-Colouring problem is an intensively studied problem, which has additionally attracted attention in its guise of the *constraint satisfaction problem* (CSP), especially since the seminal paper of Feder and Vardi [14]. Their well-known conjecture, recently proved by Bulatov [4] and Zhuk [32], stated that every CSP(H) has complexity either in P or NP-complete, omitting any Ladner-like complexities in between.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 49; pp. 49:1–49:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


Figure 1 Relations between Surjective H-Colouring and its variants (from [15]). An arrow from one problem to another indicates that the latter problem is polynomial-time solvable for a graph H whenever the former is polynomial-time solvable for H. Reverse arrows do not hold for the leftmost and rightmost arrows, as witnessed by the reflexive 4-vertex cycle for the rightmost arrow and by any reflexive tree that is not a reflexive interval graph for the leftmost arrow (Feder, Hell and Huang [11] showed that the only reflexive bi-arc graphs are reflexive interval graphs). It is not known whether the reverse direction holds for the two middle arrows.

This paper concerns the computational complexity of the *surjective homomorphism* problem, also known in the literature as Surjective H-Colouring [15, 16] and H-Vertex-Compaction [30]. This problem requires the homomorphism to be surjective. It is a cousin of the *list homomorphism* problem and is even more closely related to the *retraction* and *compaction* problems. Indeed, the H-Compaction problem, hitherto defined only for graphs H, takes as input a graph G and asks if there exists a function f from $V(\mathrm{G})$ to $V(\mathrm{H})$ so that for each non-loop edge $(x, y) \in E(\mathrm{H})$ (i.e. with $x \neq y$), there exists $u, v \in V(\mathrm{G})$ so that $f(u) = x$ and $f(v) = y$. Thus, compaction can be seen as the *edge-surjective homomorphism* problem.[1] The problem H-Retraction takes as input a superstructure G of H and asks whether there is a homomorphism from G to H that is the identity on H. The H-Retraction problem is polynomially equivalent with a special type of CSP, CSP(H′), where H′ is H decorated with constants naming the elements of its domain. Feder and Vardi [14] showed that the task of classifying the complexities of the retraction problems is equivalent to that for the CSPs. Hence, owing to [4, 32], H-Retraction has now been fully classified.

The list homomorphism problem, List H-colouring, allows one to express restricted lists for each of the input structure's elements, that are the only domain elements permitted in a solution homomorphism. List H-colouring is also a special type of CSP, CSP(H′), where H′ is H replete with all possible unary relations over the domain of H. Historically, the complexities of List H-colouring were the first to be settled by Bulatov [3], following important earlier work on graphs [9, 10, 11].

In contrast to the situation for H-Colouring, List H-Colouring and H-Retraction, the complexity classifications for H-Compaction and Surjective H-Colouring are far from settled, and there are concrete open cases (see 3-No-Rainbow-Colouring in the survey [2]). Obtaining NP-hardness for compaction and surjective homomorphism problems appears to be especially challenging. The complexity-theoretic relationship between these various problems is drawn in Figure 1. At present it is not known whether there is a graph H so that H-Retraction, H-Compaction and Surjective H-Colouring do not have the same complexity up to polynomial time reduction (see [15, 29]).

Nevertheless classification results for Surjective H-Colouring have tried to keep pace with similar ones for H-Retraction. In [12] it is proved, among partially reflexive pseudoforests H, where the problem H-Retraction splits between P and NP-complete. A similar classification for Surjective H-Colouring over partially reflexive forests can be inferred from the classification for partially reflexive trees in [16]. The quest for a classification for H-Compaction and Surjective H-Colouring over pseudoforests is ongoing, but for both problems already the reflexive 4-cycle took some time to classify [24, 27], as well as the irreflexive 6-cycle [28, 31].

[1] Except for the treatment of self-loops, which appears to be an idiosyncrasy that plays no vital role in computational complexity. For some history of the definition see [27].

The above results are for undirected graphs, whereas we focus on digraphs. A known classification for H-Retraction comes for irreflexive semicomplete digraphs H. In [1] Bang-Jensen, Hell, and MacGillivray proved that H-Colouring is always in P or is NP-complete if H is irreflexive semicomplete. This is a fortiori a classification for H-Retraction since semicomplete digraphs are *cores* (all endomorphisms are automorphisms), which ensures that H-Colouring and H-Retraction are polynomially equivalent. For irreflexive semicomplete digraphs H, the classification for Surjective H-Colouring can be read trivially from that for H-Colouring, and they are the same. An obvious next place to look is at the situation if H is *reflexive* semicomplete, where surely the classifications will not be the same as H-Colouring is trivial in this case.

Reflexive tournaments form an important subclass of the class of reflexive semicomplete graphs and are well-understood algebraically [19]. In particular, the classification for H-Retraction where H is a reflexive tournament can be inferred from the algebraic characterisation from [19]: for a reflexive tournament H, the H-Retraction problem is in NL if H is transitive, and it is NP-complete otherwise. This raises the question whether the same holds for Surjective H-Colouring and whether we can develop algebraic methods further to prove this. In fact, the algebraic method is by now well known for CSPs and their relatives, including its use with digraphs; see the recent survey [20]. However, the algebraic method is not so far advanced for surjective homomorphism problems. So far it only exists in the work of Chen [7], who proved that Surjective H-Colouring is NP-complete if H has the property that all of its polymorphisms depend only on one variable, that is, are essentially unary. Chen's result has not yet been put to work (even on toy open problems) and a key driver for our research has been to find, in the wild, a place for its application.

Our Results. We give, for the first time, complexity classifications for Surjective H-Colouring for digraphs instead of undirected graphs. To prove our results, we further develop algebraic machinery to tackle surjective homomorphism problems. That is, in Section 2 we introduce, after giving the necessary terminology, the concept of endo-triviality. We show how this concept is closely related to some known algebraic concepts and explore its algorithmic consequences in the remainder of our paper.

Firstly, in Section 3, we prove that a reflexive digraph H that is endo-trivial has the property that all of its polymorphisms are essentially unary. Combining this result with the aforementioned result of Chen [7] immediately yields that Surjective H-Colouring is NP-complete for any such digraph H. This is the first concrete application of Chen's result to settle a problem of open complexity; it shows, for instance, that Surjective H-Colouring is NP-complete if H is a reflexive directed cycle on $k \geq 3$ vertices. As the case $k \leq 2$ is trivial, this gives a classification of Surjective H-Colouring for reflexive directed cycles, which we believe form a natural class of digraphs to consider given the results in [24, 31].

Secondly, in Section 4 we give a complexity classification for Surjective H-Colouring, when H is a reflexive tournament. We use endo-triviality in an elaborate and recursive encoding of an NP-hard retraction problem within Surjective H-Colouring. In doing this, we show that on this class, the complexities of Surjective H-Colouring and H-Retraction coincide.

Finally, our results enable us to give a complexity classification for Surjective H-Colouring when H is a partially reflexive digraph of size at most 3. In doing this, we show that on this class, the complexities of Surjective H-Colouring and H-Retraction coincide. We are not aware of an existing classification for H-Retraction on this class, but we do build on one existing for List H-Colouring from [13].

2 Preliminaries

Let $[n] := \{1, \ldots, n\}$. For a k-tuple $\bar{t}$ and $i \in [k]$, let $\bar{t}[i]$ be the ith entry in $\bar{t}$. In a digraph G, a forward- (resp., backward-) *neighbour* (or *adjacent*) to a vertex $u \in V(\mathrm{G})$ is another vertex $v \in V(\mathrm{G})$ so that $(u, v) \in E(\mathrm{G})$ (resp., $(v, u) \in E(\mathrm{G})$). The *out-degree* and *in-degree* of a vertex are the number of its forward-neighbours and backward-neighbours, respectively. A vertex with out-degree and in-degree both 0 is said to be *isolated*. A vertex with a self-loop is *reflexive* and otherwise it is *irreflexive*. A digraph is *(ir)reflexive* if all its vertices are (ir)reflexive.

The *directed path* on k vertices is the digraph with vertices $u_0, \ldots, u_{k-1}$ and edges (u_i, u_{i+1}) for $i = 0, \ldots, k-2$. The *directed cycle* on k vertices is obtained from the directed path on k vertices after adding the edge (u_{k-1}, u_0). A digraph G is *strongly connected* if for all $u, v \in V(\mathrm{G})$ there is a directed path in $E(\mathrm{G})$ from u to v (note that we take this to include the situation $u = v$, but for reflexive graphs the distinction is moot). A digraph is *weakly connected* if its symmetric closure (underlying undirected graph) is connected. A *double-edge* in a digraph G consists in a pair of distinct vertices $u, v \in V(\mathrm{G})$, so that $(u, v), (v, u) \in E(\mathrm{G})$. A digraph G is *semicomplete* if for every two distinct vertices u and v, at least one of (u, v), (v, u) belongs to $E(\mathrm{G})$. A digraph G is a *tournament* if for every two distinct vertices u and v, exactly one of (u, v), (v, u) belongs to $E(\mathrm{G})$. We demand our tournaments have more than one vertex (to rule out certain trivial cases in proofs). A reflexive tournament G is *transitive* if for every triple of vertices u, v, w with $(u, v), (v, w) \in E(\mathrm{G})$, also (u, w) belongs to $E(\mathrm{G})$. A digraph F is a *subgraph* of a digraph G if $V(\mathrm{F}) \subseteq V(\mathrm{G})$ and $E(\mathrm{F}) \subseteq E(\mathrm{G})$. It is *induced* if $E(\mathrm{F})$ coincides with $E(\mathrm{G})$ restricted to pairs containing only vertices of $V(\mathrm{F})$. A *subtournament* is an induced subgraph of a tournament (note that this is a fortiori a tournament). All subgraphs we consider in this paper will be induced.

A *homomorphism* from a digraph G to a digraph H is a function $f : V(\mathrm{G}) \to V(\mathrm{H})$ so that for all $u, v \in V(\mathrm{G})$ with $(u, v) \in E(\mathrm{G})$ we have $(f(u), f(v)) \in E(\mathrm{H})$. We say that f is *(vertex)-surjective* if for every vertex $x \in V(\mathrm{H})$ there exists a vertex $u \in V(\mathrm{G})$ with $f(u) = x$. Let H be a digraph. A *homomorphic image* of H is a digraph H′ so that there is a surjective homomorphism $h : \mathrm{H} \to \mathrm{H}'$ in which, for all $(x', y') \in E(\mathrm{H}')$ there exists $(x, y) \in E(\mathrm{H})$ so that $x' = h(x)$ and $y' = h(y)$. That is, h is vertex- and edge-surjective.

The *direct product* of two digraphs G and H, denoted $\mathrm{G} \times \mathrm{H}$, has vertex set $V(\mathrm{G}) \times V(\mathrm{H})$ and edges $((x, y), (x', y'))$ exactly when $(x, x') \in E(\mathrm{G})$ and $(y, y') \in E(\mathrm{H})$. This product is associative and commutative, up to isomorphism, and spawns a natural power. A k-ary *polymorphism* of G is a function $f : \mathrm{G}^k \to \mathrm{G}$ so that when $(x_1, y_1), \ldots, (x_k, y_k) \in E(\mathrm{G})$ then $(f(x_1, \ldots, x_k), f(y_1, \ldots, y_k)) \in E(\mathrm{G})$. A polymorphism of G can be seen as a homomorphism from the kth (direct) power of G, G^k, to G. A polymorphism f is *idempotent* if for all $x \in V(\mathrm{G})$, $f(x, \ldots, x) = x$. The k-ary ith *projection*, for $i \in [k]$, is the polymorphism π_k^i given by $\pi_k^i(x_1, \ldots, x_k) = x_i$. A k-ary operation f is called *essentially unary* if there exists a unary operation g and $i \in [k]$ so that $f(x_1, \ldots, x_k) = g(x_i)$ for all $(x_1, \ldots, x_k) \in \mathrm{G}^k$.

Let G be a digraph. An *endomorphism* of G is a homomorphism from G to itself. An endomorphism e of G is a *constant map* if there exists a vertex $v \in V(\mathrm{G})$ such that $e(u) = v$ for all $u \in V(\mathrm{G})$. The *endomorphism digraph* G^{G} has as its vertices the endomorphisms of G, and there is an edge $(f, g) \in E(\mathrm{G}^{\mathrm{G}})$ between endomorphisms f and g if and only if for every edge $(x, y) \in E(\mathrm{G})$, we have that $(f(x), g(y)) \in E(\mathrm{G})$. We note that G^{G} is reflexive when G is reflexive and also make two more observations. The first one follows directly from the definition of G^{G} as well. The second one can, for example, be found in Section 5.2 of [21].

▶ **Lemma 1.** *If* $(f_1, g_1) \in E(\mathrm{G}^{\mathrm{G}})$ *and* $(f_2, g_2) \in E(\mathrm{G}^{\mathrm{G}})$*, then* $((f_1 \circ f_2), (g_1 \circ g_2)) \in E(\mathrm{G}^{\mathrm{G}})$.

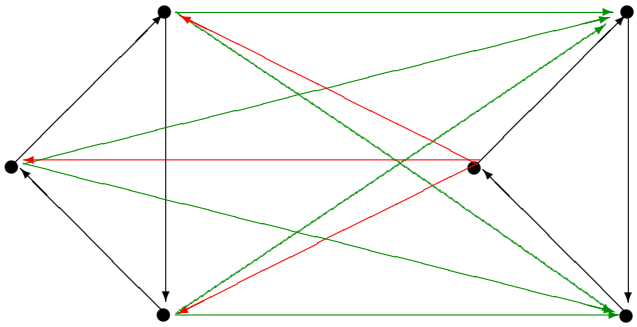


Figure 2 A tournament on six vertices (self-loops are not drawn), which retracts to the directed 3-cycle (in black) on the right-hand side, but not to the one on the left-hand side (in black as well). However, there is no endomorphism that maps the left-hand one isomorphically to the right. We can use this tournament to build a structure that is a counterexample to the generalisation of Lemma 5 stating that endo-trivial and retract-trivial coincide. Let us label the vertices in the tournament: α, β, γ (left-hand DC_3^*, clockwise from bottom) and $0, 1, 2$ (right-hand DC_3^*, clockwise from bottom). Let us build a structure B by augmenting a new 6-ary relation with tuples in $\{(\alpha, \beta, \gamma, 0, 1, 2), (\alpha, \alpha, \alpha, \alpha, \beta, \gamma), (\alpha, \alpha, \alpha, \alpha, \alpha, \alpha)\}$. The structure B is retract-trivial but is not endo-trivial, since it has an interesting endomorphism that takes $(\alpha, \beta, \gamma, 0, 1, 2)$ to $(\alpha, \alpha, \alpha, \alpha, \beta, \gamma)$.

▶ **Lemma 2.** *Let* G *and* H *be two digraphs. Let* ϕ *be a homomorphism from* $\mathrm{H} \times \mathrm{G}$ *to* G*. Then the function* ψ *defined by* $\psi(x)(u) = \phi(x, u)$ *for all* $x \in V(\mathrm{H})$*,* $u \in V(\mathrm{G})$ *is a homomorphism from* H *to* G^{G}*.*

A bijective endomorphism whose inverse is a homomorphism is an *automorphism*. An endomorphism is *non-trivial* if it is neither an automorphism nor a constant map. A digraph, all of whose endomorphisms are automorphisms, is termed a *core*. An endomorphism e of a digraph H *fixes* a subset $S \subseteq V(\mathrm{H})$ if $e(S) = S$, that is, $e(x) \in S$ for all $x \in S$, and it fixes a subgraph F of H if $e(\mathrm{F}) = \mathrm{F}$. It fixes an induced subgraph F *up to automorphism* if $e(\mathrm{F})$ is an automorphic copy of F (this is a stronger condition than $e(\mathrm{F})$ being isomorphic to F). An endomorphism r of G is a *retraction* of G if r is the identity on the image $r(\mathrm{G})$ (thus a retraction must have at least one fixed point).

Endo-triviality and Retract-triviality. We now define the key concept of endo-triviality and the closely related concept of retract-triviality.

▶ **Definition 3.** A digraph is *endo-trivial* if all of its endomorphisms are automorphisms or constant maps.

The concept of endo-triviality also arises from the perspective of the algebra of polymorphisms. An algebra is called minimal if its unary polynomials are either constants or the permutations (see Definition 2.14 in [17]). For reflexive digraphs, polynomials and polymorphisms coincide. In other words, a reflexive digraph is endo-trivial if and only if its associated algebra of polymorphisms is minimal.

We will also need the following closely related concept.

▶ **Definition 4.** A digraph is *retract-trivial* if all of its retractions are the identity or constant maps.

The concept of retract-triviality also appears in the algebraic theory but has, as far as we are aware, not been studied in a combinatorial setting. An algebra is term-minimal if the only retractions in its clone of terms are the identity and constants (see [26]). A reflexive digraph

is retract-minimal if its associated algebra of polymorphisms is term-minimal. It follows that on reflexive digraphs, the concepts of retract-minimality and retract-triviality coincide.

We note that every endo-trivial structure is also retract-trivial. However, the reverse implication is not necessarily true: in Figure 2 we give an example of a structure that is retract-trivial but not endo-trivial. This example is based on a digraph but is not itself a digraph. It is also possible to construct a retract-trivial digraph that is not endo-trivial [25], but on reflexive tournaments both concepts do coincide.

▶ **Lemma 5.** *A reflexive tournament is endo-trivial if and only if it is retract-trivial.*

Proof. (Forwards.) Trivial. (Backwards.) By contraposition, suppose e is a non-trivial endomorphism of a reflexive tournament H. Consider $e(\mathrm{H})$ and build some function e^{-1} from $e(\mathrm{H})$ to H by choosing $e^{-1}(y) = x$ if $e(x) = y$ arbitrarily. Since H is a (reflexive) tournament, e^{-1} is an isomorphism, whereupon $e^{-1} \circ e$ is the identity automorphism when restricted to some subtournament H_0 of H. Hence $e^{-1} \circ e$ is a non-trivial retraction of H (to H_0). ◀

3 Essential Unarity and a Dichotomy for Reflexive Directed Cycles

In this section we give the first concrete application, of which we are aware, of the aforementioned result of Chen, formally stated below.

▶ **Theorem 6** (Corollary 3.5 in [7])**.** *Let* H *be a finite structure whose universe* $V(\mathrm{H})$ *has size strictly greater than* 1*. If each polymorphism of* H *is essentially unary, then* Surjective H-Colouring *is* NP*-complete.*

In order to this, we make use of the endomorphism graph and a result from Máróti and Zádori [23]. Let id_{H} denote the identity map on a digraph H.

▶ **Lemma 7** (Lemma 2.2 in [23])**.** *Let* H *be a reflexive digraph. If* $(id_{\mathrm{H}}, f) \in E(\mathrm{H}^{\mathrm{H}})$*, where* f *is different from* id_{H}*, then* H *has a non-surjective retraction* r *such that* $(id_{\mathrm{H}}, r) \in E(\mathrm{H}^{\mathrm{H}})$*.*

The following lemma is crucial and will be of use in the next section as well.

▶ **Lemma 8.** *Let* H *be a retract-trivial reflexive digraph with at least three vertices. Then*

1. H *has no double edge;*
2. H *is strongly connected; and*
3. *the automorphisms of* H *are isolated vertices in* H^{H}*.*

We use Lemma 8 to obtain the following structural result.

▶ **Theorem 9.** *Let* H *be an endo-trivial reflexive digraph with at least three vertices. Then every polymorphism of* H *is essentially unary.*

Proof. Since H is endo-trivial, H is retract-trivial. Hence, by Lemma 8, H is strongly connected, and furthermore the automorphisms of H are isolated vertices of H^{H}. As H is endo-trivial, this means that H^{H} is the disjoint union of a copy of H that corresponds to the constant maps and a set of isolated vertices, one for each automorphism of H. Suppose for a contradiction that there exists an an n-ary polymorphism f of H which is not essentially unary. We may without loss of generality assume that f depends on all of its n variables, where $n \geq 2$. By Lemma 2, the mapping $F : \mathrm{H}^{n-1} \to \mathrm{H}^{\mathrm{H}}$ defined by $F(x_1, \ldots, x_{n-1})(y) = f(x_1, \ldots, x_{n-1}, y)$ is a homomorphism. Since H is strongly connected, so is H^{n-1}, and hence so is the image of F in H^{H}. Thus this image is either contained in the component of constants, in which case f does not depend on its last variable, else it is a singleton, in which case f does not depend on any of its first $n-1$ variables. ◀

Combining Theorems 6 and 9 yields the main result of this section.

▶ **Corollary 10.** *If* H *is an endo-trivial reflexive digraph on at least three vertices, then* Surjective H-Colouring *is* NP*-complete.*

Let DC_k^* denote the reflexive directed cycle on k vertices, which is readily seen to be endo-trivial. Corollary 10 yields the following dichotomy for reflexive directed cycles after noting that Surjective DC_k^*-Colouring is trivial for $k \leq 2$.

▶ **Corollary 11.** Surjective DC_k^*-Colouring *is in* L *if* $k \leq 2$ *and* NP*-complete if* $k \geq 3$.

It is not difficult to construct endo-trivial reflexive tournaments other than reflexive directed cycles. In the next section though we give a combinatorial NP-hardness proof for Surjective H-Colouring whenever H is *any* non-transitive reflexive tournament. As DC_3^* is such a digraph, this proof also can be used for the case $\mathrm{H} = \mathrm{DC}_3^*$. However, it does not extend to Surjective DC_k^*-Colouring for $k \geq 4$.

4 A Dichotomy for Reflexive Tournaments

In this section we prove our main result, namely a dichotomy of Surjective H-Colouring for reflexive tournaments H by showing that transitivity is the crucial property for tractability. In the next subsections we prove that Surjective H-Colouring is NP-complete when H is a non-transitive tournament.

4.1 Two Elementary Lemmas

It is well-known that every strongly connected tournament has a directed Hamilton cycle [6]. Hence we derive the following corollary to Lemmas 5 and 8 Part 2.

▶ **Lemma 12.** *If* H *is a reflexive tournament that is endo-trivial, then* H *contains a directed Hamilton cycle.*

We will also need the following lemma.

▶ **Lemma 13.** *If* H *is a reflexive tournament that is endo-trivial, then any homomorphic image of* H *of size* $1 < n < |V(\mathrm{H})|$ *possesses a double edge.*

Proof. Suppose H has a homomorphic image of size $1 < n < |V(\mathrm{H})|$ without a double edge. By looking at the equivalence classes of vertices identified in the homomorphic image, we can deduce a non-trivial retraction, namely by mapping each of the vertices in an equivalence class to any particular one of them. ◀

4.2 The NP-Hardness Gadget

We now introduce the gadget Cyl_m^* drawn in Figure 3. We take m disjoint copies of the directed m-cycle DC_m^* arranged in a cylindrical fashion so that there is an edge from i in the jth copy to i in the $j+1$th copy (drawn in red), and an edge from i in the $j+1$th copy to $i+1$ in the jth copy (drawn in green). We consider DC_m^* to have vertices $\{1, \ldots, m\}$. A key role will be played by Hamilton cycles HC_m in a strongly connected reflexive tournament on m vertices. We consider this cycle also labelled $\{1, \ldots, m\}$, in order to attach it to the gadget Cyl_m^*. The gadget Cyl_m^* is an alteration of a gadget that appears in [9] for proving that List H-Colouring is NP-complete when H is an undirected cycle on at least four vertices, but our proof is very different.

The following lemma follows from induction on the copies of DC_m^*, since a reflexive tournament has no double edges.

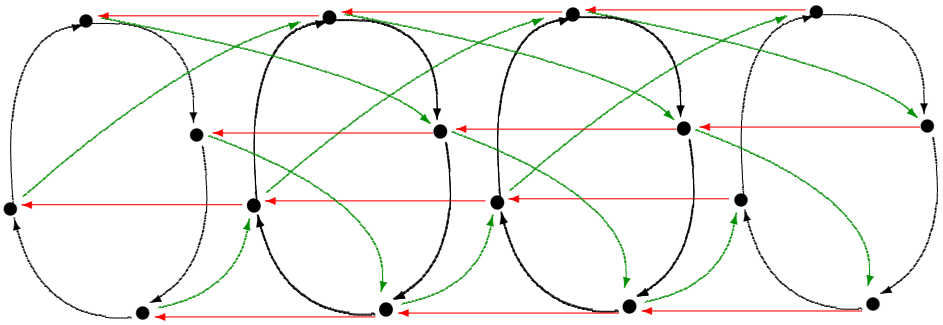


Figure 3 The gadget Cyl^*_m in the case $m := 4$ (self-loops are not drawn). We usually visualise the right-hand copy of DC^*_4 as the "bottom" copy and then we talk about vertices "above" and "below" according to the red arrows.

▶ **Lemma 14.** *In any homomorphism h from Cyl^*_m, with bottom cycle DC^*_m, to a reflexive tournament, if $|h(\mathrm{DC}^*_m)| = 1$, then $|h(\mathrm{Cyl}^*_m)| = 1$.*

We will use another property, denoted (†), of Cyl^*_m, which is that the retractions from Cyl^*_m to its bottom copy of DC^*_m, once propagated through the intermediate copies, induce on the top copy precisely the set of automorphisms of DC^*_m. That is, the top copy of DC^*_m is mapped isomorphically to the bottom copy, and all such isomorphisms may be realised. The reason is that in such a retraction, the $(j+1)$th copy may either map under the identity to the jth copy, or rotate one edge of the cycle clockwise, and Cyl^*_m consists of sufficiently many (namely m) copies of DC^*_m.

Now let H be a reflexive tournament that contains a subtournament H_0 on m vertices that is endo-trivial. By Lemma 12, we find that H_0 contains at least one directed Hamilton cycle HC_0. Define $\mathrm{Spill}_m(\mathrm{H}[\mathrm{H}_0, \mathrm{HC}_0])$ as follows. Begin with H and add a copy of the gadget Cyl^*_m, where the bottom copy of DC^*_m is identified with HC_0, to build a digraph $\mathrm{F}(\mathrm{H}_0, \mathrm{HC}_0)$. Now ask, for some $y \in V(\mathrm{H})$ whether there is a retraction r of $\mathrm{F}(\mathrm{H}_0, \mathrm{HC}_0)$ to H so that some vertex x in the top copy of DC^*_m in Cyl^*_m is such that $r(x) = y$. Such vertices y comprise the set $\mathrm{Spill}_m(\mathrm{H}[\mathrm{H}_0, \mathrm{HC}_0])$.

We now observe that $\mathrm{Spill}_m(\mathrm{H}[\mathrm{H}_0, \mathrm{HC}_0]) = V(\mathrm{H})$ if H retracts to H_0.

▶ **Lemma 15.** *If* H *is a reflexive tournament that retracts to a subtournament* H_0 *with Hamilton cycle* HC_0*, then* $\mathrm{Spill}_m(\mathrm{H}[\mathrm{H}_0, \mathrm{HC}_0]) = V(\mathrm{H})$.

4.3 Two Base Cases

Recall that if H is an endo-trivial tournament, then Surjective H-Colouring is NP-complete due to Corollary 10. However H may not be endo-trivial. We will now show how to deal with the case where H is not endo-trivial but retracts to an endo-trivial subtournament. For doing this we use the above gadget, but we need to distinguish between two different cases.

▶ **Lemma 16** (Base Case I.)**.** *Let* H *be a reflexive tournament that retracts to an endo-trivial subtournament* H_0 *with Hamilton cycle* HC_0*. Assume that* H *retracts to* H'_0 *for every isomorphic copy* $\mathrm{H}'_0 = i(\mathrm{H}_0)$ *of* H_0 *in* H *with* $\mathrm{Spill}_m(\mathrm{H}[\mathrm{H}'_0, i(\mathrm{HC}_0)]) = V(\mathrm{H})$*. Then* H_0-Retraction *can be polynomially reduced to* Surjective H-Colouring.

Proof. Let G be an instance of H_0-Retraction. We build an instance G″ of Surjective H-Colouring in the following fashion. First, take a copy of H together with G and build

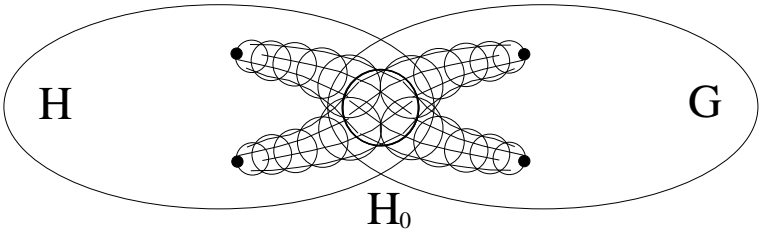


Figure 4 A stylised depiction of the construction in Base Case I. The central circle is the Hamilton cycle and the eccentric circles emanating thereout are the gadgets Cyl^*_m.

G' by identifying these on the copy of H_0 that they both possess as a subgraph. Let m be the size of H_0 and consider its Hamilton cycle HC_0. We build G'' from G' by augmenting a new copy of Cyl^*_m for every vertex $v \in V(\mathrm{G}') \setminus V(\mathrm{H}_0)$. Vertex v is to be identified with any vertex in the top copy of DC^*_m in Cyl^*_m and the bottom copy of DC^*_m is to be identified with HC_0 in H_0 according to the identity function. See Figure 4 for an example. We claim that G retracts to H_0 if and only if there exists a surjective homomorphism from G'' to H.

First suppose that G retracts to H_0. Let h be a retraction from G to H_0. We extend h as follows. First we map the copy of H in G'' to itself in H by the identity. This will ensure surjectivity. We then map the various copies of Cyl^*_m in G''. This is always possible: because H retracts to H_0, we have $\mathrm{Spill}_m(\mathrm{H}[\mathrm{H}_0, \mathrm{HC}_0]) = V(\mathrm{H})$ due to Lemma 15. Hence, if $h(x) = y$ for two vertices $x \in V(\mathrm{G}') \setminus V(\mathrm{H}_0)$ and $y \in V(\mathrm{H})$, we can always find a retraction of the graph $\mathrm{F}(\mathrm{H}_0, \mathrm{HC}_0)$ to H that maps x to y, and we mimic this retraction on the corresponding subgraph in G''. The crucial observation is that this can be done independently for each vertex in $V(\mathrm{G}') \setminus V(\mathrm{H}_0)$, as two vertices of different copies of Cyl^*_m are only adjacent if they both belong to G'. This leads to a surjective homomorphism from G'' to H.

Now suppose that there exists a surjective homomorphism h from G'' to H. If $|h(\mathrm{H}_0)| = 1$, then by Lemma 14, $|h(\mathrm{Cyl}^*_m)| = 1$ for all copies of Cyl^*_m in G''. This means that $|h(\mathrm{G}'')| = 1$ and h is not surjective, a contradiction. Now, $1 < |h(\mathrm{H}_0)| < m$ is not possible either due to Lemma 13. Thus, $|h(\mathrm{H}_0)| = m$ and indeed h maps H_0 to a copy of itself in H which we will call $\mathrm{H}'_0 = i(\mathrm{H}_0)$ for some isomorphism i.

We claim that $\mathrm{Spill}_m(\mathrm{H}[\mathrm{H}'_0, i(\mathrm{HC}_0)]) = V(\mathrm{H})$. In order to see this, consider a vertex $y \in V(\mathrm{H})$. As h is surjective, there exists a vertex $x \in V(\mathrm{G}'')$ with $h(x) = y$. By construction, x belongs to some copy of DC^*_m, and thus also belongs to some copy of DC^*_m in $\mathrm{F}(\mathrm{H}_0, \mathrm{HC}_0)$. We can extend i^{-1} to an isomorphism from the copy of Cyl^*_m (which has $i(\mathrm{HC}_0)$ as its bottom cycle) in the graph $\mathrm{F}(\mathrm{H}'_0, i(\mathrm{HC}_0))$ to the copy of Cyl^*_m (which has HC_0 as its bottom cycle) in the graph $\mathrm{F}(\mathrm{H}_0, \mathrm{HC}_0)$. We define a mapping r^* from $\mathrm{F}(\mathrm{H}'_0, i(\mathrm{HC}_0))$ to H by $r^*(u) = h \circ i^{-1}(u)$ if u is on the copy of Cyl^*_m in $\mathrm{F}(\mathrm{H}'_0, i(\mathrm{HC}_0))$ and $r^*(u) = u$ otherwise. We observe that $r^*(u) = u$ if $u \in V(\mathrm{H}'_0)$ as h coincides with i on H_0. As H_0 separates the other vertices of the copy of Cyl^*_m from $V(\mathrm{H}) \setminus V(\mathrm{H}_0)$, in the sense that removing H_0 would disconnect them, this means that r^* is a retraction from $\mathrm{F}(\mathrm{H}'_0, i(\mathrm{HC}_0))$ to H. We find that r^* maps $i(x)$ to $h \circ i^{-1}(i(x)) = h(x) = y$. Moreover, as x is in some copy of DC^*_m in $\mathrm{F}(\mathrm{H}_0, \mathrm{HC}_0)$, we have that $i(x)$ is in some copy of DC^*_m in $\mathrm{F}(\mathrm{H}'_0, i(\mathrm{HC}_0))$. We may assume without loss of generality that $i(x)$ belongs to the top copy. We conclude that y always belongs to $\mathrm{Spill}_m(\mathrm{H}[\mathrm{H}'_0, i(\mathrm{HC}_0)])$.

As $\mathrm{Spill}_m(\mathrm{H}[\mathrm{H}'_0, i(\mathrm{HC}_0)]) = V(\mathrm{H})$, we find, by assumption of the lemma, that there exists a retraction r from H to H'_0. Now $i^{-1} \circ r \circ h$ ia the desired retraction of G to H_0. ◀

We now need to deal with the situation in which we have an isomorphic copy $\mathrm{H}'_0 = i(\mathrm{H}_0)$ of H_0 in H with $\mathrm{Spill}_m(\mathrm{H}[\mathrm{H}'_0, i(\mathrm{HC}_0)]) = V(\mathrm{H})$, such that H does not retract to H'_0 (see

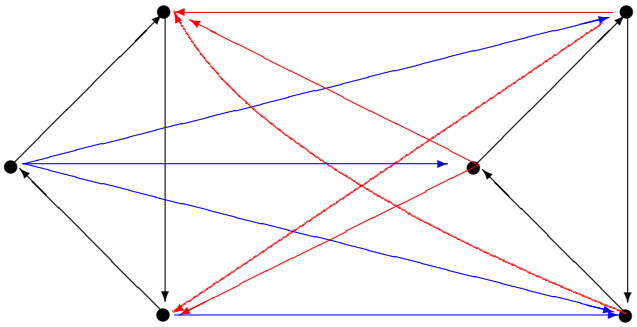


Figure 5 An interesting tournament H on six vertices (self-loops are not drawn). This tournament does not retract to the DC_3^* on the left-hand side, yet $\mathrm{Spill}_3(\mathrm{H}[\mathrm{DC}_3^*, \mathrm{DC}_3]) = V(\mathrm{H})$.

Figure 5 for an example). We cannot deal with this case in a direct matter and first show another base case. For this we need the following lemma and an extension of endo-triviality that we discuss afterwards.

▶ **Lemma 17.** *Let* H *be a reflexive tournament, containing a subtournament* H_0 *so that any endomorphism of* H *that fixes* H_0 *is an automorphism. Then any endomorphism of* H *that maps* H_0 *to an isomorphic copy* $\mathrm{H}_0' = i(\mathrm{H}_0)$ *of itself is an automorphism of* H*.*

Proof. For contradiction, suppose there is an endomorphism h that maps H_0 to an isomorphic copy $\mathrm{H}_0' = i(\mathrm{H}_0)$ of itself that is not an automorphism of H. In particular, $|h(\mathrm{H})| < |V(\mathrm{H})|$. Choose h^{-1} in the following fashion. We let h^{-1} of $h(\mathrm{H}_0)$ be the natural isomorphism of $h(\mathrm{H}_0)$ to H_0 (that inverts the isomorphism given by h from H_0 to H_0'). Otherwise we choose h^{-1} arbitrarily, such that $h^{-1}(y) = x$ only if $h(x) = y$. Since H is a reflexive tournament, containing precisely one edge between distinct vertices, h^{-1} is an isomorphism. Moreover, $h^{-1} \circ h$ is an endomorphism of H that fixes H_0 and that is not an automorphism, a contradiction. ◀

Let H_0 be an induced subgraph of a digraph H. We say that the pair $(\mathrm{H}, \mathrm{H}_0)$ is *endo-trivial* if all endomorphisms of H that fix H_0 are automorphisms.

▶ **Lemma 18** (Base Case II)**.** *Let* H *be a reflexive tournament with a subtournament* H_0 *with Hamilton cycle* HC_0 *so that* $(\mathrm{H}, \mathrm{H}_0)$ *and* H_0 *are endo-trivial and* $\mathrm{Spill}_m(\mathrm{H}[\mathrm{H}_0, \mathrm{HC}_0]) = V(\mathrm{H})$*. Then* H-Retraction *can be polynomially reduced to* Surjective H-Colouring*.*

4.4 Generalising the Base Cases

We now generalise the two base cases to more general cases via some recursive procedure. Afterwards we will show how to combine these two cases to complete our proof. We will first need a slightly generalised version of Lemma 17, which nonetheless has virtually the same proof.

▶ **Lemma 19.** *Let* $\mathrm{H}_2 \supset \mathrm{H}_1 \supset H_0$ *be a sequence of strongly connected reflexive tournaments, each one a subtournament of the one before. Suppose that any endomorphism of* H_1 *that fixes* H_0 *is an automorphism. Then any endomorphism* h *of* H_2 *that maps* H_0 *to an isomorphic copy* $\mathrm{H}_0' = i(\mathrm{H}_0)$ *of itself also gives an isomorphic copy of* H_1 *in* $h(\mathrm{H}_1)$*.*

The following two lemmas generalize Lemmas 16 and 18.

▶ **Lemma 20** (General Case I). *Let* $\mathrm{H}_0, \mathrm{H}_1, \ldots, \mathrm{H}_k, \mathrm{H}_{k+1}$ *be reflexive tournaments, the first* k *of which have Hamilton cycles* $\mathrm{HC}_0, \mathrm{HC}_1, \ldots, \mathrm{HC}_k$*, respectively, so that* $\mathrm{H}_0 \subseteq H_1 \subseteq \cdots \subseteq \mathrm{H}_k \subseteq \mathrm{H}_{k+1}$*. Assume that* H_0*,* $(\mathrm{H}_1, \mathrm{H}_0)$*, . . . ,* $(\mathrm{H}_k, \mathrm{H}_{k-1})$ *are endo-trivial and that*

$$\begin{array}{lcl} \mathrm{Spill}_{a_0}(\mathrm{H}_1[\mathrm{H}_0, \mathrm{HC}_0]) & = & V(\mathrm{H}_1) \\ \mathrm{Spill}_{a_1}(\mathrm{H}_2[\mathrm{H}_1, \mathrm{HC}_1]) & = & V(\mathrm{H}_2) \\ \vdots & \vdots & \vdots \\ \mathrm{Spill}_{a_{k-1}}(\mathrm{H}_k[\mathrm{H}_{k-1}, \mathrm{HC}_{k-1}]) & = & V(\mathrm{H}_k). \end{array}$$

Assume that H_{k+1} *retracts to* H_k *and also to every isomorphic copy* $\mathrm{H}'_k = i(\mathrm{H}_k)$ *of* H_k *in* H_{k+1} *with* $\mathrm{Spill}_{a_k}(\mathrm{H}_{k+1}[\mathrm{H}'_k, i(\mathrm{HC}_k)]) = V(\mathrm{H}_{k+1})$*. Then* H_k-Retraction *can be polynomially reduced to* Surjective H_{k+1}-Colouring*.*

▶ **Lemma 21** (General Case II). *Let* $\mathrm{H}_0, \mathrm{H}_1, \ldots, \mathrm{H}_k, \mathrm{H}_{k+1}$ *be reflexive tournaments, the first* $k+1$ *of which have Hamilton cycles* $\mathrm{HC}_0, \mathrm{HC}_1, \ldots, \mathrm{HC}_k$*, respectively, so that* $\mathrm{H}_0 \subseteq H_1 \subseteq \cdots \subseteq \mathrm{H}_k \subseteq \mathrm{H}_{k+1}$*. Suppose that* H_0*,* $(\mathrm{H}_1, \mathrm{H}_0)$*, . . . ,* $(\mathrm{H}_k, \mathrm{H}_{k-1}), (\mathrm{H}_{k+1}, \mathrm{H}_k)$ *are endo-trivial and that*

$$\begin{array}{lcl} \mathrm{Spill}_{a_0}(\mathrm{H}_1[\mathrm{H}_0, \mathrm{HC}_0]) & = & V(\mathrm{H}_1) \\ \mathrm{Spill}_{a_1}(\mathrm{H}_2[\mathrm{H}_1, \mathrm{HC}_1]) & = & V(\mathrm{H}_2) \\ \vdots & \vdots & \vdots \\ \mathrm{Spill}_{a_{k-1}}(\mathrm{H}_k[\mathrm{H}_{k-1}, \mathrm{HC}_{k-1}]) & = & V(\mathrm{H}_k) \\ \mathrm{Spill}_{a_k}(\mathrm{H}_{k+1}[\mathrm{H}_k, \mathrm{HC}_k]) & = & V(\mathrm{H}_{k+1}) \end{array}$$

Then H_{k+1}-Retraction *can be polynomially reduced to* Surjective H_{k+1}-Colouring*.*

4.5 Final Steps for Hardness for Non-Transitive Reflexive Tournaments

We first prove, by using the lemmas from Section 4.4, that Surjective H-Colouring is NP-complete if H is a non-transitive reflexive tournament that is strongly connected. For our discourse it is not necessary to know precisely what is a Taylor operation, but we will use the following result.

▶ **Theorem 22** ([5, 22]). *Let* H *be a finite structure so that the idempotent polymorphisms of* H *omit all Taylor operations. Then* H-Retraction *is* NP*-complete.*

▶ **Corollary 23.** *Let* H *be a strongly connected reflexive tournament. Then* Surjective H-Colouring *is* NP*-complete.*

In order to deal with reflexive tournaments that are not strongly connected we need the following strengthened version of Corollary 23.

▶ **Corollary 24.** *Let* H *be a strongly connected reflexive tournament. Then* Surjective H-Colouring *is* NP*-complete even for strongly connected digraphs.*

We now give our main hardness result following with our main dichotomy.

▶ **Theorem 25.** *Let* H *be a non-transitive reflexive tournament. Then* Surjective H-Colouring *is* NP*-complete.*

▶ **Corollary 26.** *Let* H *be a reflexive tournament. If* H *is transitive, then* Surjective H-Colouring *is in* NL*; otherwise it is* NP*-complete.*

Proof. For the transitive case we can say that H-Retraction is in NL from [8], since H enjoys the ternary median operation as a polymorphism (this has been observed, inter alia, in [19]). It follows of course that Surjective H-Colouring is in NL also. The non-transitive case follows from Theorem 25. ◀

5 Digraphs with a most three vertices

▶ **Theorem 27.** *Let* H *be a partially reflexive digraph of size at most* 3*. Then* Surjective H-Colouring *is polynomially equivalent to* H-Retraction*. In particular, it is always in* P *or is* NP*-complete.*

6 Conclusion

We have given the first significant classification results for Surjective H-Colouring where H comes from a class of digraphs (that are not graphs). To do this, we have developed both a novel algebraic method and a novel recursive combinatorial method. Below we discuss some directions for future research.

Let 3NRC be the hypergraph with vertex-set $\{r, g, b\}$ and hyperedge-set $\{r, g, b\} \setminus \{(r, g, b), (r, b, g), (g, b, r), (g, r, b), (b, r, g), (b, g, r)\}$. Then 3-No-Rainbow-Colouring is the problem Surjective 3NRC-Colouring, in which one looks for a surjective colouring of the vertices, such that no hyperedge is rainbow-coloured (i.e. uses all colours). We recall that the complexity of this problem is open since it arose (under a different name) in [18], see also Question 3 in [2]. The Surjective DC_3^*-Colouring problem is the digraph problem most closely related to 3-No-Rainbow-Colouring. To explain this, when looking for digraphs with a similar character to 3NRC, we would insist at least that the automorphism group is transitive. This leaves just the reflexive and irreflexive directed 3-cycles and the reflexive and irreflexive 3-cliques, that is, 3-cycles with a double edge between every pair of vertices (admittedly, the cycles have only some of the automorphisms of the cliques). If H is the reflexive 3-clique, then H-Retraction and Surjective H-Colouring are trivial. If H is the irreflexive directed 3-cycle, then H has a majority polymorphism, which shows that H-Retraction, and thus Surjective H-Colouring (see Figure 1), can be solved in polynomial time [1]. If H is the irreflexive 3-clique, then Surjective H-Colouring is NP-complete, as there exists a straightforward reduction from 3-Colouring. Hence $H = DC_3^*$ was indeed the only case for which determining the complexity of Surjective H-Colouring was not immediately obvious.

It would be great to extend our results to larger reflexive digraph classes. Reflexive digraphs with a double edge are not endo-trivial and further fail to be endo-trivial in the worse way, since Surjective DC_2^*-Colouring is nearly trivial. Thus, our methods are likely only to be applicable to reflexive oriented digraphs, that is, those without a double edge. On the way, a natural question arising is exactly which reflexive digraphs are endo-trivial?

Finally, there is the question as to whether the assumption of endo-triviality can be weakened to that of retract-triviality in Theorem 9. Endo-triviality is used right at the beginning of the proof to show that G^G is the disjoint union of a copy of G (the constant maps) and isolated automorphisms. We do not know if retract-triviality is here sufficient.

References

1 Jørgen Bang-Jensen, Pavol Hell, and Gary MacGillivray. The complexity of colouring by semicomplete digraphs. *SIAM Journal on Discrete Mathematics*, 1(3):281–298, 1988.

2 Manuel Bodirsky, Jan Kára, and Barnaby Martin. The complexity of surjective homomorphism problems - a survey. *Discrete Applied Mathematics*, 160(12):1680–1690, 2012.

3 Andrei A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Transactions on Computational Logic*, 12(4):24:1–24:66, 2011.

4 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. *Proc. FOCS 2017*, pages 319–330, 2017.

5 Andrei A. Bulatov, Peter Jeavons, and Andrei A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005.

6 Paul Camion. Chemins et circuits hamiltoniens de graphes complets. *C. R. Acad. Sci. Paris*, 249:2151–2152, 1959.

7 Hubie Chen. An algebraic hardness criterion for surjective constraint satisfaction. *Algebra Universalis*, 72(4):393–401, 2014.

8 Víctor Dalmau and Andrei A. Krokhin. Majority constraints have bounded pathwidth duality. *European Journal of Combinatorics*, 29(4):821–837, 2008.

9 Tomás Feder and Pavol Hell. List homomorphisms to reflexive graphs. *Journal of Combinatorial Theory, Series B*, 72(2):236–250, 1998.

10 Tomás Feder, Pavol Hell, and Jing Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19(4):487–505, 1999.

11 Tomás Feder, Pavol Hell, and Jing Huang. Bi-arc graphs and the complexity of list homomorphisms. *Journal of Graph Theory*, 42(1):61–80, 2003.

12 Tomás Feder, Pavol Hell, Peter Jonsson, Andrei A. Krokhin, and Gustav Nordh. Retractions to pseudoforests. *SIAM Journal on Discrete Mathematics*, 24(1):101–112, 2010.

13 Tomás Feder, Pavol Hell, and Kim Tucker-Nally. Digraph matrix partitions and trigraph homomorphisms. *Discrete Applied Mathematics*, 154(17):2458–2469, 2006.

14 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, 1998.

15 Petr A. Golovach, Matthew Johnson, Barnaby Martin, Daniël Paulusma, and Anthony Stewart. Surjective H-colouring: New hardness results. *Proc. CiE 2017, Lecture Notes in Computer Science*, 10307:270–281, 2017.

16 Petr A. Golovach, Daniël Paulusma, and Jian Song. Computing vertex-surjective homomorphisms to partially reflexive trees. *Theoretical Computer Science*, 457:86–100, 2012.

17 David Charles Hobby and Ralph McKenzie. *The Structure of Finite Algebras, Contemporary Mathematics*. American Mathematical Society, 1988.

18 Daniel Král, Jan Kratochvíl, Andrzej Proskurowski, and Heinz-Jürgen Voss. Coloring mixed hypertrees. *Discrete Applied Mathematics*, 154(4):660–672, 2006.

19 Benoit Larose. Taylor operations on finite reflexive structures. *International Journal of Mathematics and Computer Science*, 1(1):1–26, 2006.

20 Benoit Larose. Algebra and the complexity of digraph CSPs: a survey. In Andrei A. Krokhin and Stanislav Zivny, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 267–285. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

21 Benoit Larose, Cynthia Loten, and Claude Tardif. A characterisation of first-order constraint satisfaction problems. *Logical Methods in Computer Science*, 3(4), 2007.

22 Benoit Larose and László Zádori. Finite posets and topological spaces in locally finite varieties. *Algebra Universalis*, 52(2):119–136, 2005.

23 Miklós Maróti and László Zádori. Reflexive digraphs with near unanimity polymorphisms. *Discrete Mathematics*, 312(15):2316–2328, 2012.

24 Barnaby Martin and Daniël Paulusma. The computational complexity of disconnected cut and 2_{k2}-partition. *Journal of Combinatorial Theory, Series B*, 111:17–37, 2015.

25 Mark Siggers. personal communication.

26 Agnes Szendrei. Term minimal algebras. *Algebra Universalis*, 32(4):439–477, 1994.

27 Narayan Vikas. Computational complexity of compaction to reflexive cycles. *SIAM Journal on Computing*, 32(1):253–280, 2002.

28 Narayan Vikas. Computational complexity of compaction to irreflexive cycles. *Journal of Computer and System Sciences*, 68(3):473–496, 2004.

29 Narayan Vikas. A complete and equal computational complexity classification of compaction and retraction to all graphs with at most four vertices and some general results. *Journal of Computer and System Sciences*, 71(4):406–439, 2005.

30 Narayan Vikas. Algorithms for partition of some class of graphs under compaction and vertex-compaction. *Algorithmica*, 67(2):180–206, 2013.

31 Narayan Vikas. Computational complexity of graph partition under vertex-compaction to an irreflexive hexagon. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, volume 83 of *LIPIcs*, pages 69:1–69:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

32 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. *Proc. FOCS 2017*, pages 331–342, 2017.

Pumping Lemmas for Weighted Automata

Filip Mazowiecki
University of Oxford, Oxford, UK

Cristian Riveros
Pontificia Universidad Católica de Chile, Santiago, Chile

Abstract

We present three pumping lemmas for three classes of functions definable by fragments of weighted automata over the min-plus semiring and the semiring of natural numbers. As a corollary we show that the hierarchy of functions definable by unambiguous, finitely-ambiguous, polynomially-ambiguous weighted automata, and the full class of weighted automata is strict for the min-plus semiring.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Weighted Automata, Regular Functions over Words, Pumping Lemmas

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.50

Funding The first author was partially supported by EPSRC grants EP/M011801/1 and EP/M012298/1. The last author was supported by FONDECYT grant 11150653 and the Millennium Nucleus Center for Semantic Web Research under grant NC120004.

Acknowledgements We thank Shaull Almagor and the anonymous referees for their helpful comments.

1 Introduction

Weighted automata (WA) are an expressible extension of finite state automata for computing functions over words. They have been extensively studied since Schützenberger [28], and its decidability problems [18, 1], extensions [9], logic characterization [9, 17], and applications [22, 7] have been deeply investigated.

The class of functions defined by WA has several equivalent representations in terms of computational models or logics. Recently Alur et al. introduced the computational model of cost register automata (CRA) [2, 3], an alternative model for computing functions over words, which are currently extensively studied [20, 21, 8]. The idea of this model is to enhance deterministic finite automata with registers that can be combined by using operations over a fixed semiring. In [2], it was shown that CRA are strictly more expressive than WA. Interestingly, it was also shown that a natural fragment of CRA is equally expressive to WA, which gives a new representation to understand this class of functions.

Regarding the logical representation of WA, Droste and Gastin introduced in [9] the so-called Weighted Logics (WL), a natural extension of monadic second order logics (MSO) from the boolean semiring to any commutative semiring. The semantics of this logics maps any formula in MSO over strings to one or zero in the semiring, depending whether the input satisfies the formula or not. Furthermore, WL includes sum and product quantifiers that allow to aggregate the output of boolean formulas producing an output value in the semiring. Although WL is far more expressive than WA, it was shown in [9] that a natural syntactic restriction of WL is equally expressive to WA, giving the first logical characterization of WA.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 50; pp. 50:1–50:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


Weighted logics or, more generally, quantitative logics have found many applications in understanding WA [10, 17], verification [5] and computational complexity [4].

The complexities of decision problems for WA have also been investigated, unfortunately often with undecidability results [18, 1]. For this reason various fragments of WA over different semirings have been studied. Recently, over one-letter alphabets, where WA are equivalent to linear recurrences, some new decidability results were shown for limited fragments [23, 24]. Other restrictions of WA involve bounding their numbers of runs. Among them most studied classes are *unambiguous automata*, *finitely-ambiguous automata*, and *polynomially-ambiguous automata*, where the numbers of accepting runs is bounded by 1, a constant, a polynomial in the size of input, respectively [29, 16, 15]. These are robust subclasses of functions inside WA that also have found several characterization in terms of cost register automata [2] and weighted logics [17].

Although functions defined by WA and its subclasses have been studied in terms of representations and decidability, little is known about its expressibility. Indeed, we are not aware of any general techniques to show if a function is definable or not by WA or any of its subclasses. Results related to the inexpressibility of WA usually require sophisticated arguments for each particular function [16, 20] and there is no clear path to generalize these techniques. As a matter of fact, the strict inclusions between unambiguous, finitely-ambiguous, polynomially-ambiguous, and the full class of WA are "well-known" to the community, but it is hard to find references to formal proofs (see related work below). In contrast, for regular languages or first order logics there exist elegant and useful techniques for showing inexpressibility like, for example, the standard pumping lemma for regular languages [13] or Ehrenfeucht-Fraïssé games for first-order logics [12, 11, 19]. One would like to have similar techniques in the quantitative world that simplifies inexpressibility arguments of WA, cost register automata, or even weighted logics to a small number of lines. Such techniques help to understand the inner structure of these functions and unveil their limits of expressibility.

In this paper, we embark in the work of loading the expressibility toolbox of weighted automata with pumping lemmas. We present three pumping lemmas, each of them for a different class or subclass of functions defined by WA over the min-plus semiring or the semiring of natural numbers. For every pumping lemma we show examples of functions that do not satisfy the lemma, giving very short inexpressibility proofs. Our results do not attempt to fully characterize the class or subclasses of weighted automata in terms of pumping properties, nor to provide conditions that can be verified by a computer. Our goal is to give the first tools for expressibility of weighted automata and to provide researchers with simple arguments for showing that functions do not belong to a given class.

Related work. In [14] it is shown that over the min-plus semiring polynomially-ambiguous automata are strictly more expressive than finite-ambiguous automata. In [16] strict inclusions between unambiguous automata, finitely-ambiguous automata, and the full class of WA are shown over the max-plus semiring. In both papers the strict inclusions are shown by analyzing particular functions. Using results in [6] one can deduce that unambiguous automata are strictly included in the other classes over the min-plus and max-plus semirings. Gathering these results we obtain strict inclusions between unambiguous automata, finitely-ambiguous automata, and the full class of WA over the min-plus semiring. However, to our knowledge, there is no reference for a strict inclusion between polynomially-ambiguous automata and the full class of WA.

Organization. In Section 2 we introduce weighted automata and some basic definitions. In Section 3 and Section 4 we present and prove pumping lemmas for weighted automata over the semiring of natural numbers and its extension using the operation min. In Section 5 we show the pumping lemma for polynomially-ambiguous automata over the min-plus semiring.

Some concluding remarks can be found in Section 6.

2 Preliminaries

In this section, we recall the definitions of weighted automata (WA). We start with the definitions that are standard in this area. A monoid $\mathbb{M} = (M, \otimes, \mathbb{1})$ is a set M with an associative operation $\otimes$ and a neutral element $\mathbb{1}$. Standard examples of monoids are: the set of words Σ^* with concatenation and empty word; or the set of matrices with multiplication and the identity matrix. A semiring is a structure $\mathbb{S} = (S, \oplus, \odot, \mathbb{0}, \mathbb{1})$, where $(S, \oplus, \mathbb{0})$ is a commutative monoid, $(S - \{\mathbb{0}\}, \odot, \mathbb{1})$ is a monoid, multiplication distributes over addition, and $\mathbb{0} \odot s = s \odot \mathbb{0} = \mathbb{0}$ for each $s \in S$. If the multiplication is commutative, we say that $\mathbb{S}$ is commutative. In this paper, we always assume that $\mathbb{S}$ is commutative. We usually denote S or M by the name of the semiring or monoid $\mathbb{S}$ or $\mathbb{M}$. In this paper, we are interested in the *min-plus semiring* $(\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$ and the *semiring of natural numbers with* ∞ $(\mathbb{N} \cup \{\infty\}, +, \cdot, 0, 1)$ where we assume that $\infty + n = \infty$ for every $n \in \mathbb{N} \cup \{\infty\}$ and $\infty \cdot n = \infty$ if $n \neq 0$ and 0 otherwise. We denote the former by $\mathbb{N}_{\min,+}$ and the later by $\mathbb{N}_{+,\times}$. Note that $\mathbb{N}_{+,\times}$ is an extension of the standard semiring of natural numbers $\mathbb{N}$ and all our results for $\mathbb{N}_{+,\times}$ also hold for $\mathbb{N}$. We use this extended version of $\mathbb{N}$ to easily apply some results from $\mathbb{N}_{+,\times}$ to $\mathbb{N}_{\min,+}$ (see Section 4). Given a finite set Q, we denote by $\mathbb{S}^{Q \times Q}$ ($\mathbb{S}^Q$) the set of square matrices (vectors resp.) over $\mathbb{S}$ indexed by Q. The algebra induced by $\mathbb{S}$ over $\mathbb{S}^{Q \times Q}$ and $\mathbb{S}^Q$ is defined as usual.

We also consider two finite semirings that will be useful during proofs. We consider the boolean semiring $\mathbb{B} = (\{0,1\}, \vee, \wedge, 0, 1)$ and the extended boolean semiring $\mathbb{B}_\infty = (\{0, 1, \infty\}, \vee, \wedge, 0, 1)$ such that $\infty \vee n = \infty$ for every $n \in \{0, 1, \infty\}$, $\infty \wedge 0 = 0$, and $\infty \wedge n = \infty$ if $n \in \{1, \infty\}$. Both finite semirings will be used as *abstractions* of $\mathbb{N}_{\min,+}$ and $\mathbb{N}_{+,\times}$, respectively.

In this paper, we study the specification of functions from words to values, namely, from Σ^* to $\mathbb{S}$. We say that a function $f : \Sigma^* \to \mathbb{S}$ is definable by a computational system $\mathcal{A}$ (e.g. by WA) if $f(w) = [\![\mathcal{A}]\!](w)$ for any $w \in \Sigma^*$, where $[\![\mathcal{A}]\!]$ is the semantics of $\mathcal{A}$ over words.

2.1 Weighted automata

Fix a finite alphabet Σ and a commutative semiring $\mathbb{S}$. A *weighted automaton* (WA) over Σ and $\mathbb{S}$ is a tuple $\mathcal{A} = (Q, \Sigma, \{M_a\}_{a \in \Sigma}, I, F)$ where Q is a finite set of states, $\{M_a\}_{a \in \Sigma}$ is a set of matrices such that $M_a \in \mathbb{S}^{Q \times Q}$ and $I, F \in \mathbb{S}^Q$ are the initial and the final vectors, respectively [27, 10]. We say that a state q is initial if $I(q) \neq \mathbb{0}$ and accepting if $F(q) \neq \mathbb{0}$. We usually say that an entry $M_a(p,q) = s$ is a transition and write $p \xrightarrow{a/s} q$. Furthermore, we say that a run ρ of $\mathcal{A}$ over a word $w = a_1 \dots a_n$ is a sequence of transitions: $\rho = q_0 \xrightarrow{a_1/s_1} q_1 \xrightarrow{a_2/s_2} \cdots \xrightarrow{a_n/s_n} q_n$, where $s_i \neq \mathbb{0}$ for all $1 \leq i \leq n$ and $I(q_0) \neq \mathbb{0}$. We refer to q_i as the i-th state of the run ρ. The run ρ is accepting if $F(q_n) \neq \mathbb{0}$, and the weight of an accepting run ρ is defined by $|\rho| = I(q_0) \odot (\bigodot_{i=1}^n s_i) \odot F(q_n)$. We define $\mathrm{Run}_{\mathcal{A}}(w)$ as the set of all accepting runs of $\mathcal{A}$ over w. Finally, the output of $\mathcal{A}$ over a word w is defined by $[\![\mathcal{A}]\!](w) = I^t \cdot M_{a_1} \cdot \ldots \cdot M_{a_n} \cdot F = \bigoplus_{\rho \in \mathrm{Run}_{\mathcal{A}}(w)} |\rho|$ where I^t is the transpose of I and the second sum is equal to $\mathbb{0}$ if $\mathrm{Run}_{\mathcal{A}}(w)$ is empty. For a word $w = a_1 \dots a_n$ we usually denote $M_w = M_{a_1} \cdot \ldots \cdot M_{a_n}$ and then $[\![\mathcal{A}]\!](w) = I^t \cdot M_w \cdot F$. Note that $M_w(p,q)$ provides the cost of moving from state p to state q reading the word w.

A weighted automaton $\mathcal{A}$ is called *unambiguous* (U-WA) if $|\mathrm{Run}_{\mathcal{A}}(w)| \leq 1$ for every $w \in \Sigma^*$; and $\mathcal{A}$ is called *finitely-ambiguous* (FA-WA) if there exists a uniform bound N such that $|\mathrm{Run}_{\mathcal{A}}(w)| \leq N$ for every $w \in \Sigma^*$ [29, 16]. Furthermore, $\mathcal{A}$ is called *polynomially-ambiguous* (PA-WA) if the function $|\mathrm{Run}_{\mathcal{A}}(w)|$ is bounded by a polynomial in the length of w [15]. We

call classes of functions definable by such automata *unambiguous regular*, *finitely-ambiguous regular* and *polynomially-ambiguous regular* functions. The class of functions defined by weighted automata are called regular functions.

Note that every unambiguous WA over $\mathbb{N}_{\min,+}$ can be defined by a polynomially-ambiguous WA over $\mathbb{N}_{+,\times}$ [16, 2] (recall that ∞ is in $\mathbb{N}_{+,\times}$). Therefore, the class of unambiguous regular functions over $\mathbb{N}_{\min,+}$ is included in the class of regular functions over $\mathbb{N}_{+,\times}$ (see Example 1). This inclusion is strict since regular functions over $\mathbb{N}_{\min,+}$ are always bounded by a linear function in the size of the word, and it is easy to define the function $f(w) = 2^{|w|}$ over $\mathbb{N}_{+,\times}$. Below, we give several examples of functions defined by WA over $\mathbb{N}_{+,\times}$ and $\mathbb{N}_{\min,+}$ that will be used in paper. Recall that in the latter semiring $\nvdash = \infty$ and $\odot = +$. Transitions $p \xrightarrow{a/s} q$, where $s = \nvdash$ are omitted.

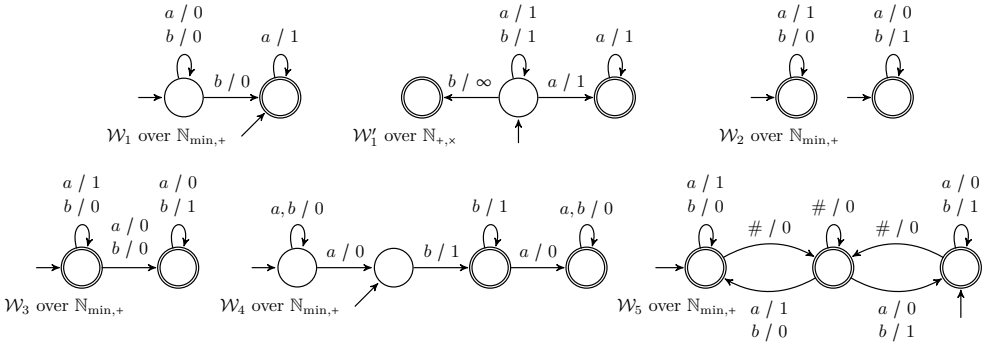


Figure 1 Examples of weighted automata. For WA over $\mathbb{N}_{\min,+}$ the initial and accepting states are labeled by 0 in the corresponding vector, and ∞ otherwise. Similarly, for WA over $\mathbb{N}_{+,\times}$ the initial and accepting states are labeled by 1 in the corresponding vector, and 0 otherwise.

▶ **Example 1.** Let $\Sigma = \{a, b\}$. Consider the function f_1 that for given word $w \in \Sigma^*$ outputs the length of the biggest suffix of a's (and ∞ if the word ends in b). This is defined by $\mathcal{W}_1$ over $\mathbb{N}_{\min,+}$ in Figure 1. One can easily check that $\mathcal{W}_1$ is unambiguous, hence f_1 belongs to unambiguous regular functions over $\mathbb{N}_{\min,+}$. In Figure 1, $\mathcal{W}_1'$ over $\mathbb{N}_{+,\times}$ also defines f_1.

▶ **Example 2.** Let $\Sigma = \{a, b\}$. Consider the function f_2 that for given word $w \in \Sigma^*$ outputs $\min\{|w|_a, |w|_b\}$, namely, counts the number of each letter and returns the minimum. This is defined by $\mathcal{W}_2$ in Figure 1. The WA $\mathcal{W}_2$ is finitel-ambiguous, hence f_2 belongs to finitely-ambiguous regular functions.

▶ **Example 3.** Let $\Sigma = \{a, b\}$. Consider the function f_3 that for a given word $w = a_1 \dots a_n \in \Sigma^*$ outputs $\min_{0 \le i \le n}\{|a_1 \dots a_i|_a + |a_{i+1} \dots a_n|_b\}$. This is defined by $\mathcal{W}_3$ in Figure 1. The WA is polynomially-ambiguous, hence f_3 belongs to polynomially-ambiguous functions.

▶ **Example 4.** Let $\Sigma = \{a, b\}$. Consider the function f_4 that for a given word $w \in \Sigma^*$ computes the shortest subword of b's (if there is none it outputs ∞). This is defined by $\mathcal{W}_4$ in Figure 1. The WA is polynomially-ambiguous, hence f_4 belongs to polynomially-ambiguous functions.

▶ **Example 5.** Let $\Sigma = \{a, b, \#\}$. Consider the function f_5 such that, for any $w \in \Sigma^*$ of the form $w_0 \# w_1 \# \dots \# w_n$ with $w_i \in \{a, b\}^*$, it computes the minimum number of a's or b's for each subword w_i (i.e. $\min\{|w_i|_a, |w_i|_b\}$) and then it sums these values over all subwords w_i, that is, $f_5(w) = \sum_{i=0}^{n} \min\{|w_i|_a, |w_i|_b\}$. This is defined by $\mathcal{W}_5$ in Figure 1. Given that the WA has an exponential number of runs, the function f_5 is a regular function but not necessarily a polynomially-ambiguous regular function.

We assume that our weighted automata are always trim, namely, all their states are reachable from some initial state (i.e., they are accessible) and they can reach some final state (i.e., they are co-accessible). Verifying if a state is accessible or co-accessible is reduced to a reachability test in the transition graph [25] and this can be done in NLogSpace. Thus, we can assume without loss of generality that all our automata are trimmed.

2.2 Finite monoids and idempotents

We say that a monoid is finite if the set of its elements is finite. Let $\mathbb{M} = (M, \otimes, \mathbb{1})$ be a finite monoid. We say that $\iota \in \mathbb{M}$ is an idempotent if $\iota \otimes \iota = \iota$. The following lemma is a standard result for finite monoids and idempotents (e.g. see Theorem 6.37 in [26]).

▶ **Lemma 6.** *Let $\mathbb{M}$ be a finite monoid. There exists $N > 0$ such that for every sequence $m_1 \otimes \ldots \otimes m_n$ with $m_i \in \mathbb{M}$ and $n \geq N$, there exist a factorization:*

$$(m_1 \otimes \ldots \otimes m_i) \otimes (m_{i+1} \otimes \ldots \otimes m_j) \otimes (m_{j+1} \ldots \otimes m_n),$$

where $i < j \leq n$ and $(m_{i+1} \otimes \ldots \otimes m_j)$ is an idempotent.

We will work with the finite monoid of matrices $\mathbb{B}^{Q\times Q}$ or $\mathbb{B}_\infty^{Q\times Q}$. For this, we define abstractions, i.e., homomorphisms of $\mathbb{N}_{\min,+}^{Q\times Q}$ to $\mathbb{B}^{Q\times Q}$ and $\mathbb{N}_{+,\times}^{Q\times Q}$ to $\mathbb{B}_\infty^{Q\times Q}$. These are given by the homomorphisms defined on elements of the matrices $h_1 : \mathbb{N}_{\min,+} \to \mathbb{B}$ and $h_2 : \mathbb{N}_{+,\times} \to \mathbb{B}_\infty$, defined: $h_1(m) = 0$ iff $m = \infty$; and $h_2(m) = 0$ if $m = 0$, $h_2(m) = \infty$ if $m = \infty$ and $h_2(m) = 1$ otherwise. For matrices $M \in \mathbb{N}_{\min,+}^{Q\times Q}$ or $N \in \mathbb{N}_{+,\times}^{Q\times Q}$ we denote by $\bar{M} = h_1(M)$ or $\bar{N} = h_2(N)$ their abstractions in $\mathbb{B}^{Q\times Q}$ or $\mathbb{B}_\infty^{Q\times Q}$, respectively.

3 Regular functions without min

In this section we consider regular functions over $\mathbb{N}_{+,\times}$. As a corollary of the pumping lemma in this section we show that FA-WA are strictly more expressive than U-WA over $\mathbb{N}_{\min,+}$ (Example 8). Moreover, we show that there are finitely-ambiguous regular functions over $\mathbb{N}_{\min,+}$ that cannot be defined by any regular function over $\mathbb{N}_{+,\times}$.

We introduce some notation to simplify the presentation. Given $u \cdot v \cdot w = \hat{u} \cdot \hat{v} \cdot \hat{w}$, where $u, v, w, \hat{u}, \hat{v}, \hat{w} \in \Sigma^*$, we say that $\hat{u} \cdot \underline{\hat{v}} \cdot \hat{w}$ is a *refinement* of $u \cdot \underline{v} \cdot w$ if there exist u', w' such that $u \cdot u' = \hat{u}$, $w' \cdot w = \hat{w}$, $u' \cdot \hat{v} \cdot w' = v$, and $\hat{v} \neq \epsilon$. We underline the infixes v and $\hat{v}$ to emphasize the refined part.

▶ **Theorem 7** (Pumping Lemma for regular functions over $\mathbb{N}_{+,\times}$)**.** *Let $f : \Sigma^* \to \mathbb{N} \cup \{\infty\}$ be a regular function over $\mathbb{N}_{+,\times}$. There exists N such that for all words of the form $u \cdot v \cdot w \in \Sigma^*$ with $|v| \geq N$, there exists a refinement $\hat{u} \cdot \underline{\hat{v}} \cdot \hat{w}$ of $u \cdot \underline{v} \cdot w$ such that at least one of the following two conditions holds:*

- $f(\hat{u} \cdot \underline{\hat{v}}^i \cdot \hat{w}) = f(\hat{u} \cdot \underline{\hat{v}}^{i+1} \cdot \hat{w})$ *for every $i \geq N$.*
- $f(\hat{u} \cdot \underline{\hat{v}}^i \cdot \hat{w}) < f(\hat{u} \cdot \underline{\hat{v}}^{i+1} \cdot \hat{w})$ *for every $i \geq N$.*

Before going into the details of the proof let us show how to use the lemma.

▶ **Example 8.** We show that f_2 from Example 2 is not definable by any WA over $\mathbb{N}_{+,\times}$. Indeed, suppose it is definable and fix N from Theorem 7. Consider the word $w = a^{(N+1)^2}\underline{b^N}$ and notice that $f_2(w) = N$. By refining w we get $\hat{u} \cdot \underline{\hat{v}} \cdot \hat{w} = a^{(N+1)^2} b^n \underline{b^m} b^l$ for some n, m, l such that $1 \leq m \leq N$ and $n + m + l = N$. Since $n + m \cdot N + l < n + m \cdot (N+1) + l < (N+1)^2$ it must be the case that $f_2(\hat{u} \cdot \underline{\hat{v}}^i \cdot \hat{w}) < f_2(\hat{u} \cdot \underline{\hat{v}}^{i+1} \cdot \hat{w})$ for all $i \geq N$. However, $f_2(\hat{u} \cdot \underline{\hat{v}}^i \cdot \hat{w}) = (N+1)^2$ for i sufficiently large, which is a contradiction.

▶ **Example 9.** On the other hand, the function f_1 from Example 1 satisfies Theorem 7. Consider a word $u \cdot \underline{v} \cdot w \in \Sigma^*$ and its refinement $\hat{u} \cdot \underline{\hat{v}} \cdot \hat{w}$. If $\hat{w}$ or $\hat{v}$ contain b then $f(\hat{u} \cdot \underline{\hat{v}}^i \cdot \hat{w}) = f(\hat{u} \cdot \underline{\hat{v}}^{i+1} \cdot \hat{w})$ because the suffix of a's remains the same. Otherwise, $f(\hat{u} \cdot \underline{\hat{v}}^i \cdot \hat{w}) < f(\hat{u} \cdot \underline{\hat{v}}^{i+1} \cdot \hat{w})$ since the suffix of a's increases when pumping. Moreover, it is straightforward to generalize this argument and prove Theorem 7 for all U-WA over $\mathbb{N}_{\min,+}$.

To prove Theorem 7 we use the following definitions. For a matrix $M \in \mathbb{N}_{+,\times}^{Q\times Q}$ recall that $\bar{M}$ is its homomorphic image in $\mathbb{B}_\infty^{Q\times Q}$ (see Section 2.2). We write that M and N in $\mathbb{N}_{+,\times}^{Q\times Q}$ are equivalent, denoted $M \equiv_{\mathbb{B}_\infty} N$, iff $\bar{M} = \bar{N}$. We also extend the homomorphic image and equivalence relation from matrices to vectors. We say that $D \in \mathbb{N}_{+,\times}^{Q\times Q}$ is an *idempotent* if $\bar{D}$ is an idempotent in the finite monoid $\mathbb{B}_\infty^{Q\times Q}$.

▶ **Lemma 10.** *If $M \equiv_{\mathbb{B}_\infty} N$, then $x^T \cdot M \cdot y > 0$ if and only if $x^T \cdot N \cdot y > 0$ for every $x, y \in \mathbb{N}_{+,\times}^Q$.*

Proof. Suppose that $x^T \cdot M \cdot y > 0$. By definition $x^T \cdot M \cdot y = \sum_{p,q} x(p) \cdot M(p,q) \cdot y(q)$. Then there exist $p, q \in Q$ such that $x(p) \cdot M(p,q) \cdot y(q) > 0$ and, in particular, $M(p,q) > 0$. Given that $M \equiv_{\mathbb{B}_\infty} N$ we conclude $N(p,q) > 0$ and $x(p) \cdot N(p,q) \cdot y(q) > 0$, which proves $x^T \cdot N \cdot y > 0$. ◀

Proof of Theorem 7. Let $\mathcal{A} = (Q, \Sigma, \{M_a\}_{a\in\Sigma}, I, F)$ be a WA over $\mathbb{N}_{+,\times}$ such that $f = [\![\mathcal{A}]\!]$. Without loss of generality, we assume that $I(q) \neq \infty$ and $M_a(p,q) \neq \infty$ for every $p, q \in Q$ and $a \in \Sigma$, namely, ∞ can only appear in the final vector F. Indeed, if ∞ is used in I or some M_a, we can construct two weighted automata $\mathcal{A}', \mathcal{A}^\infty$ such that $\mathcal{A}'$ is the same as $\mathcal{A}$ but each ∞-initial state or each ∞-transition is replaced with 0, and $\mathcal{A}^\infty$ outputs ∞ if there exists some run in $\mathcal{A}$ that outputs ∞ and 0 otherwise. Note that $\mathcal{A}'$ has no ∞-transition or ∞-initial state and $\mathcal{A}^\infty$ can be constructed in such a way that only the final vector contains ∞-values. The disjoint union of $\mathcal{A}'$ and $\mathcal{A}^\infty$ is equivalent to $\mathcal{A}$.

Let $N = \max\{|Q|, K\}$ where K is the constant from Lemma 6 for the finite monoid $\mathbb{B}_\infty^{Q\times Q}$. For every word $u \cdot v \cdot w \in \Sigma^*$ such that $v = a_1 \dots a_n$ with $n \geq N$, consider the output $I^T \cdot M_u \cdot M_v \cdot M_w \cdot F$ of $\mathcal{A}$ over $u \cdot v \cdot w$. By Lemma 6, there exists a factorization of the form:

$$M_v = (M_{a_1} \cdot \ldots \cdot M_{a_i}) \cdot (M_{a_{i+1}} \cdot \cdots \cdot M_{a_j}) \cdot (M_{a_{j+1}} \cdot \ldots \cdot M_{a_n})$$

for some $i < j$ where $M_{a_{i+1}} \cdot \ldots \cdot M_{a_j}$ is an idempotent (i.e., $\bar{M}_{a_{i+1}} \cdot \ldots \cdot \bar{M}_{a_j}$ is an idempotent). We define the refinement $\hat{u} \cdot \underline{\hat{v}} \cdot \hat{w}$ of $u \cdot \underline{v} \cdot w$ such that $\hat{u} = u \cdot (a_1 \dots a_i)$, $\hat{v} = a_{i+1} \dots a_j$, and $\hat{w} = (a_{j+1} \dots a_n) \cdot w$. Furthermore, define $x = I \cdot M_u \cdot M_{a_1} \cdot \ldots \cdot M_{a_i}$, $D = M_{a_{i+1}} \cdot \ldots \cdot M_{a_j}$, and $y = M_{a_{j+1}} \cdot \ldots \cdot M_{a_n} \cdot M_w \cdot F$. Note that $f(\hat{u} \cdot \hat{v}^i \cdot \hat{w}) = x^T \cdot D^i \cdot y$ for every $i \geq 0$ and D is an idempotent (i.e. $\bar{D}$ is an idempotent). It remains to show the following lemma.

▶ **Lemma 11.** *For every idempotent $D \in \mathbb{N}_{+,\times}^{Q\times Q}$ and $x, y \in \mathbb{N}_{+,\times}^Q$ where D and x do not contain ∞-values, one of the conditions holds:*

$$x^T \cdot D^i \cdot y = x^T \cdot D^{i+1} \cdot y \quad \text{for every } i \geq |Q|, \quad \text{or} \tag{1}$$

$$x^T \cdot D^i \cdot y < x^T \cdot D^{i+1} \cdot y \quad \text{for every } i \geq |Q|. \tag{2}$$

We start showing that Lemma 11 holds when $y = e_p$ for some $p \in Q$, where $e_p(q) = 1$ if $q = p$ and 0 otherwise. Note that $z = \sum_{p\in Q} z(p) \cdot e_p$ for every vector z.

We say that p is *D-stable* (or just *stable*) if $D(p,p) > 0$. Note that if p is stable, then $D^i(p,p) > 0$ for every $i > 0$ (recall that D is idempotent). Furthermore, $D \cdot e_p = e_p + z$ for some $z \in \mathbb{N}_{+,\times}^Q$. Suppose that p is stable and $D \cdot e_p = e_p + z$ for some vector z. Then for $i > 0$:

$$x^T \cdot D^{i+1} \cdot e_p = x^T \cdot D^i \cdot (e_p + z) = x^T \cdot D^i \cdot e_p + x^T \cdot D^i \cdot z$$

Given that D is idempotent and $D^i \equiv_{\mathbb{B}_\infty} D$, by Lemma 10 we have that $x^T \cdot D^i \cdot z > 0$ if, and only if, $x^T \cdot D \cdot z > 0$. Therefore, if $x^T \cdot D \cdot z > 0$, we get that $x^T \cdot D^i \cdot e_p < x^T \cdot D^{i+1} \cdot e_p$ for every $i > 0$, in particular, for every $i \geq |Q|$. Otherwise, $x^T \cdot D \cdot z = 0$ and $x^T \cdot D^i \cdot e_p = x^T \cdot D^{i+1} \cdot e_p$ for every $i > 0$, in particular, for every $i \geq |Q|$.

Let $P \subseteq Q$ be the set of all non-stable states in D. Consider the relation $\preceq_D \subseteq P \times P$ such that $p \preceq_D q$ if $p = q$ or $D(p,q) > 0$. One can easily check that $\preceq_D$ forms a partial order over P, namely, that $\preceq_D$ is reflexive, antisymmetric, and transitive. Indeed, transitivity holds because D is idempotent. To prove antisymmetry, note that for every non-stable states p and q, if $p \preceq_D q$, $q \preceq_D p$ and $p \neq q$ hold, then $D(p,p) > 0$. This is a contradiction since p is non-stable.

Since $\preceq_D$ is a partial order, we prove the lemma for $y = e_p$ by induction over $\preceq_D$. Formally, we strengthen the inductive hypothesis such that conditions (1) and (2) hold for every $i \geq N_q$, where $N_q = |\{q' \in P \mid q' \preceq_D q\}|$ (notice that $N_q \leq |Q|$ for every q). The base case is for $N_p = 0$, which means that p is stable. In the inductive case $N_p > 0$ the state p is non-stable. Then

$$x^T \cdot D^{i+1} \cdot e_p \;=\; x^T \cdot D^i \cdot (c_1 \cdot e_{q_1} + \ldots + c_k \cdot e_{q_k}) \;=\; c_1(x^T \cdot D^i \cdot e_{q_1}) + \ldots + c_k(x^T \cdot D^i \cdot e_{q_k})$$

for pairwise different states $q_1, \ldots, q_k$ and positive values $c_1, \ldots, c_k \in \mathbb{N}$ such that q_j is either stable or $q_j \prec_D p$. Thus all states $q_1, \ldots, q_k$ satisfy our inductive hypothesis.

Consider the partition of $q_1, \ldots, q_k$ into sets $C_=$ and $C_<$ such that $C_=$ and $C_<$ satisfy condition (1) and (2), respectively. If $C_< = \varnothing$, then for every $i \geq N_p$ we have:

$$\begin{aligned} x^T \cdot D^{i+1} \cdot e_p &= c_1(x^T \cdot D^i \cdot e_{q_1}) + \ldots + c_k(x^T \cdot D^i \cdot e_{q_k}) \\ &= c_1(x^T \cdot D^{i-1} \cdot e_{q_1}) + \ldots + c_k(x^T \cdot D^{i-1} \cdot e_{q_k}) \\ &= x^T \cdot D^i \cdot e_p. \end{aligned} \tag{3}$$

Note that $x^T \cdot D^i \cdot e_{q_j} = x^T \cdot D^{i-1} \cdot e_{q_j}$ holds by the inductive hypothesis and because $N_p > N_{q_j}$ for every q_j. Suppose otherwise, that $C_< \neq \varnothing$ and there exists a state q_j that satisfies $x^T \cdot D^i \cdot e_{q_j} < x^T \cdot D^{i+1} \cdot e_{q_j}$ for every $i \geq N_{q_j}$. Then it is straightforward that equality (3) becomes a strict inequality and condition (2) holds.

We have shown that either (1) or (2) holds for $y = e_p$. It remains to extend this to any vector $y \in \mathbb{N}^Q_{+,\times}$ (possibly with ∞). Note that

$$x^T \cdot D^{i+1} \cdot y \;=\; y(q_1) \cdot (x^T \cdot D^{i+1} \cdot e_{q_1}) + \ldots + y(q_k) \cdot (x^T \cdot D^{i+1} \cdot e_{q_k})$$

for some states $q_1, \ldots, q_k$ such that $y(q_j) > 0$ for every $j \leq k$. We consider two cases. First, if there exists j such that $y(q_j) = \infty$ and $x^T \cdot D^i \cdot e_{q_j} > 0$ for $i \geq N$, then $x^T \cdot D^i \cdot y = \infty$ for every $i \geq 0$. Thus, $x^T \cdot D^i \cdot y$ satisfies condition (1). Second, suppose that for every j we have $y(q_j) \neq \infty$ or $x^T \cdot D^i \cdot e_{q_j} = 0$ for $i \geq N$. It suffices to consider the case when $y(q_j) \neq \infty$ for all j. Then if some $x^T \cdot D^i \cdot e_{q_j}$ satisfies condition (2) we have that $x^T \cdot D^i \cdot y$ satisfies condition (2). Conversely, if every $x^T \cdot D^i \cdot e_{q_j}$ satisfies condition (1) we have that $x^T \cdot D^i \cdot y$ satisfies condition (1). ◀

One could try to simplify Theorem 7 changing the condition $i \geq N$ to $i \geq 0$. Unfortunately, we do not know if the theorem would remain true. A naive approach would be to use a generalization of Lemma 6, but intuitively, the behavior of non-stable registers is problematic. Examples of this behavior are very technical and we leave this for future work. We conclude with the following remarks, straightforward from the proof. We will use them in Section 4.

▶ **Remark 12.** *Changing y to y' such that $y \equiv_{\mathbb{B}_\infty} y'$ does not influence whether condition* (1) *or condition* (2) *holds in Lemma 11 (notice that here we need that the abstractions have values in $\mathbb{B}_\infty$ not in $\mathbb{B}$). Similarly, changing x to x' such that $x \equiv_{\mathbb{B}_\infty} x'$ does not influence whether condition* (1) *or* (2) *holds.*

▶ **Remark 13.** *The constant N and the refinement of w depend only on the finite monoid $\mathbb{B}_\infty^{Q\times Q}$. In particular they are independent from the initial vectors I and F.*

4 Finite-min regular functions

In this section we focus on regular functions over $\mathbb{N}_{+,\times}$ with some min allowed. Formally, we say that $f : \Sigma^* \to \mathbb{N} \cup \{\infty\}$ is a finite-min regular function, if there exist regular functions $f_1, \ldots, f_m$ over $\mathbb{N}_{+,\times}$ such that $f(w) = \min\{f_1(w), \ldots, f_m(w)\}$. It is known that FA-WA are equivalent to a finite sum of U-WA [29], hence functions defined by FA-WA over $\mathbb{N}_{\min,+}$ are included in the class of finite-min regular functions. As a corollary of the pumping lemma in this section we show that PA-WA are strictly more expressive than FA-WA over $\mathbb{N}_{\min,+}$ (Example 15 and Example 16).

We start by introducing some notation to ease the presentation. For every word w we define an n-pumping representation

$$w = u_0 \cdot \underline{v_1} \cdot u_1 \cdot \underline{v_2} \cdot \ldots u_{n-1} \cdot \underline{v_n} \cdot u_n,$$

where $w = u_0 \cdot v_1 \cdot u_1 \cdot v_2 \cdot \ldots v_n \cdot u_n$ and $v_k \neq \epsilon$ for all k. We define a refinement of an n-pumping representation as

$$w = u'_0 \cdot \underline{y_1} \cdot u'_1 \cdot \underline{y_2} \cdot \ldots u'_{n-1} \cdot \underline{y_n} \cdot u'_n,$$

if $v_k = x_k \cdot y_k \cdot z_k$, $u'_k = z_k \cdot u_k \cdot x_{k+1}$; where $z_0 = x_{n+1} = \epsilon$ and $y_k \neq \epsilon$ for every k. Let $S \subseteq \{1, \ldots, n\}$ such that $S \neq \varnothing$. Let $\underline{v_k}$ be a fragment of an n-pumping representation w. By $\underline{v_k}(S, i)$ we denote the word v_k^i if $k \in S$ and v_k otherwise. By $w(S, i)$ we denote the word

$$w = u_0 \cdot \underline{v_1}(S,i) \cdot u_1 \cdot \underline{v_2}(S,i) \cdot \ldots u_{n-1} \cdot \underline{v_n}(S,i) \cdot u_n.$$

In other words we pump the fragments v_k for all $k \in S$.

▶ **Theorem 14** (Pumping Lemma for finite-min regular functions)**.** *Let $f : \Sigma^* \to \mathbb{N} \cup \{\infty\}$ be a finite-min regular function. There exists N such that for all n-pumping representations*

$$w = u_0 \cdot \underline{v_1} \cdot u_1 \cdot \underline{v_2} \cdot \ldots u_{n-1} \cdot \underline{v_n} \cdot u_n,$$

where $n \geq N$ and $|v_i| \geq N$ for all i, there exists a refinement

$$w = u'_0 \cdot \underline{y_1} \cdot u'_1 \cdot \underline{y_2} \cdot \ldots u'_{n-1} \cdot \underline{y_n} \cdot u'_n,$$

such that for every sequence of nonempty pairwise different subsets $S_1, \ldots, S_k \subseteq \{1 \ldots n\}$ with $k \geq N$ at least one of the following holds:

- *there exists j such that $f(w(S_j, i)) < f(w(S_j, i+1))$ for i sufficiently large;*
- *there exists $j_1 \neq j_2$ such that $f(w(S_{j_1} \cup S_{j_2}, i)) = f(w(S_{j_1} \cup S_{j_2}, i+1))$ for i sufficiently large.*

Before proving Theorem 14, we show how to use it with two examples.

▶ **Example 15.** We show that f_3 from Example 3 is not definable by finite-min regular functions. Indeed, fix N from Theorem 14 and consider the n-pumping representation $w = (\underline{b^N} \cdot \underline{a^N})^N$. We index each pumping fragment with a pair (s, j), where $j \leq N$ denotes the block and $s \leq 2$ denotes the fragment in the block. First, notice that $f_3(w) = N \cdot (N-1)$ because runs minimizing the value for $\mathcal{W}_3$ change the state after reading the last b in one of the blocks. We define the sets $S_j = \{(1, j), (2, j)\}$ for $j \in \{1, \ldots, N\}$. Clearly $f_3(w(S_j, i)) = N \cdot (N-1)$ for any j and i, because the run minimizing the value changes the state after the last b in the j-th block. On the other hand $f_3(w(S_{j_1} \cup S_{j_2}, i)) < f_3(w(S_{j_1} \cup S_{j_2}, i+1))$ for all i and $j_1 \neq j_2$. Hence f_3 does not satisfy the pumping lemma for finite-min regular functions.

▶ **Example 16.** We show that f_4 from Example 4 is not definable by finite-min regular functions. Indeed, fix N from Theorem 14. Consider the N-pumping representation $w = (\underline{b^N}a)^N$. Then by definition $f_4(w) = N$. In the refinement all pumping parts will be of the form b^n for $1 \leq n \leq N$. We define the sets $S_j = \{1,\ldots,N\} \setminus \{j\}$ for all $1 \leq i \leq N$. Clearly $f_4(w(S_j,i)) = N$ for any j and any i. On the other hand $f_4(w(S_{j_1} \cup S_{j_2},i)) < f_4(w(S_{j_1} \cup S_{j_2},i+1))$ for all i and $j_1 \neq j_2$. Hence f_4 does not satisfy the pumping lemma for finite-min regular functions.

Proof of Theorem 14. Let $f_1,\ldots,f_m$ be regular functions over $\mathbb{N}_{+,\times}$ such that $f(w) = \min\{f_1(w),\ldots,f_m(w)\}$ for every w. Furthermore, consider $\mathcal{A}_j = (Q_j,\Sigma,\{M_{j,a}\}_{a\in\Sigma},I_j,F_j)$ the corresponding WA for f_j. Let $Q = \bigcup_j Q_j$ (we assume that $Q_1,\ldots,Q_m$ are pairwise disjoint) and consider the set of matrices $\{U_a\}_{a\in\Sigma}$ where $U_a \in \mathbb{N}_{+,\times}^{Q\times Q}$ such that $U_a(p,q) = M_{j,a}(p,q)$ whenever $p,q \in Q_j$ and 0 otherwise. Then $f_j(w) = (I'_j)^t \cdot U_w \cdot F'_j$ for every j and $w \in \Sigma^*$ where I'_j and F'_j are the extensions of I_j and F_j from Q_j into Q such that $I'_j(q) = I_j(q)$ and $F'_j(q) = F_j(q)$ whenever $q \in Q_j$ and 0 otherwise. Notice that $\{U_a\}_{a\in\Sigma}$ synchronize the behavior of $f_1,\ldots,f_m$ in a single set of matrices and project the output of f_j with I'_j and F'_j. Let $N = \max\{K,m+1\}$ such that K is the constant from Lemma 6 applied to $\mathbb{B}_\infty^{Q\times Q}$. Let $w = u_0 \cdot \underline{v_1} \cdot u_1 \cdot \underline{v_2} \cdot \ldots u_{n-1} \cdot \underline{v_n} \cdot u_n$. For every v_i we use Theorem 7 over $u_{\leq i} \cdot v_i \cdot s_{\geq i}$, where $u_{\leq i} = u_0 \cdot v_1 \cdot \ldots u_{i-1}$ and $s_{\geq i} = u_i \cdot v_{i+1} \cdot \ldots u_n$ obtaining a refinement

$$w \;=\; u'_0 \cdot \underline{y_1} \cdot u'_1 \cdot \underline{y_2} \cdot \ldots u'_{n-1} \cdot \underline{y_n} \cdot u'_n,$$

where each y_i comes from Theorem 7 applied to $\{U_a\}_{a\in\Sigma}$. Recall that the refinement of $u_{\leq i} \cdot v_i \cdot s_{\geq i}$ depends only on $\{U_a\}_{a\in\Sigma}$ and not on the initial final vector (Remark 13). In particular, the refinement is the same for each function f_j. Then

$$f_j(w) \;=\; (I'_j)^t \cdot U_{u'_0} \cdot D_1 \cdot \ldots \cdot U_{u'_{n-1}} \cdot D_n \cdot U_{u'_n} \cdot F'_j$$

where $D_i = U_{y_i}$ are idempotents.

▶ **Lemma 17.** *Let $S \subseteq \{1,\ldots,n\}$ be a nonempty set and fix one function f_j. Then $f_j(w(S,i)) < f_j(w(S,i+1))$ for every $i \geq N$ iff there exists $k \in S$ such that $f_j(w(\{k\},i)) < f_j(w(\{k\},i+1))$ for every $i \geq N$.*

Proof. By definition $f_j(w(S,i)) = (I'_j)^t \cdot U_{u'_0} \cdot D_1^{s_1} \cdot \ldots \cdot U_{u'_{n-1}} \cdot D_n^{s_n} \cdot U_{u'_n} \cdot F'_j$ where $s_k = i$ if $k \in S$ and $s_k = 1$ otherwise. Since all D_i are idempotents then for all k the fragments before and after $D_k^{s_k}$ are $\equiv_{\mathbb{B}_\infty}$ equivalent, i.e.,

$$\begin{aligned}(I'_j)^t \cdot U_{u'_0} \cdot D_1^{s_1} \cdot \ldots \cdot D_{k-1}^{s_{k-1}} \cdot U_{u'_{k-1}} \;&\equiv_{\mathbb{B}_\infty}\; (I'_j)^t \cdot U_{u'_0} \cdot D_1 \cdot \ldots \cdot D_{k-1} \cdot U_{u'_{k-1}} \\ U_{u'_k} \cdot D_{k+1}^{s_{k+1}} \cdot \ldots \cdot D_n^{s_n} \cdot U_{u'_n} \cdot F'_j \;&\equiv_{\mathbb{B}_\infty}\; U_{u'_k} \cdot D_{k+1} \cdot \ldots \cdot D_n \cdot U_{u'_n} \cdot F'_j.\end{aligned}$$

Hence, the lemma follows from Remark 12. ◀

To finish the proof we analyze $f(w(S,i)) = \min\{f_1(w(S,i)),\ldots,f_m(w(S,i))\}$. Consider a sequence of subsets $S_1,\ldots,S_k$ with $k \geq N$. Suppose there is a set S_l for some l such that for every $j \leq m$ there exists $k \in S_l$ such that $f_j(w(\{k\},i)) < f_j(w(\{k\},i+1))$ for every $i \geq N$. It follows from Lemma 17 that $f(w(S_l,i)) < f(w(S_l,i+1))$ for all $i \geq N$, namely, the first condition of the theorem holds. Suppose otherwise, and for every S_l let $X_l \subseteq \{1,\ldots,m\}$ be the set of functions such that $f_j(w(S_l,i)) = f_j(w(S_l,i+1))$ for all $j \in X_l$ and $i \geq N$. Since $k \geq N > m$ there exists l_1,l_2 such that $X_{l_1} \cap X_{l_2} \neq \emptyset$. From Lemma 17 it follows that for $i \geq N$ holds: $f_j(w(S_{l_1} \cup S_{l_2},i)) = f_j(w(S_{l_1} \cup S_{l_2},i+1))$ for all $j \in X_{l_1} \cap X_{l_2}$; and $f_j(w(S_{l_1} \cup S_{l_2},i)) < f_j(w(S_{l_1} \cup S_{l_2},i+1))$ for all $j \in \{1,\ldots,m\} \setminus (X_{l_1} \cap X_{l_2})$. Hence for i sufficiently large $f(w(S_{l_1} \cup S_{l_2},i)) = \min_{j\in X_{l_1}\cap X_{l_2}}(f_j(w(S_{l_1} \cup S_{l_2},i)))$, which concludes the proof. ◀

5 Poly-ambiguous regular functions over the min-plus semiring

In this section we focus on polynomially-ambiguous regular functions over $\mathbb{N}_{\min,+}$. We expect that there is a wider class of functions, definable like in the previous section, where Theorem 18 holds but it is left for future work. A corollary from the pumping lemma in this section is that WA are strictly more expressive than PA-WA (Example 19 and 20).

We will use the notation of n-pumping representations from Section 4. As usual, a sequence of non-empty sets $S_1, \ldots, S_m$ over $\{1, \ldots, n\}$ is a *partition* if they are pairwise disjoint and $\bigcup S_i = \{1, \ldots, n\}$. Furthermore, we say that $S \subseteq \{1, \ldots, n\}$ is a *selection set* of $S_1, \ldots, S_m$ if $|S \cap S_i| = 1$ for every i.

▶ **Theorem 18** (Pumping Lemma for polynomially-ambiguous automata). *Let $f : \Sigma^* \to \mathbb{N} \cup \{\infty\}$ be a polynomially-ambiguous regular function over $\mathbb{N}_{\min,+}$. There exists N and a function $\varphi : \mathbb{N} \to \mathbb{N}$ such that for all n-pumping representations:*

$$w = u_0 \cdot \underline{v_1} \cdot u_1 \cdot \underline{v_2} \cdot \ldots \cdot u_{n-1} \cdot \underline{v_n} \cdot u_n,$$

where $|v_i| \geq N$ for every $i \leq n$, there exists a refinement:

$$w = u'_0 \cdot \underline{y_1} \cdot u'_1 \cdot \underline{y_2} \cdot \ldots u'_{n-1} \cdot \underline{y_n} \cdot u'_n,$$

such that for every partition $\pi = S_1, \ldots, S_m$ of $\{1, \ldots, n\}$ with $m \geq \varphi(\max_i(|S_i|))$, at least one of the following holds:

- *there exists j such that $f(w(S_j, i)) = f(w(S_j, i+1))$ for i sufficiently large;*
- *there exists a selection set S of π such that $f(w(S, i)) < f(w(S, i+1))$ for i sufficiently large.*

▶ **Example 19.** We show that f_5 from Example 5 is not definable by PA-WA. Indeed, let N and φ be the constant and the function from Theorem 18. Consider the following n-pumping representation: $w = (\underline{a^N} \cdot \underline{b^N}\#)^m$ where $m \geq \varphi(2)$ (here $\max_i(|S_i|) = 2$). We index each pumping fragment with a pair (s, j), where $j \leq m$ denotes the block and $s \leq 2$ denotes the fragment in the block. We define the subsets $S_1 \ldots S_m$ as follows: $S_j = \{(1, j), (2, j)\}$. Clearly for all j we have $f_5(w(S_j, i)) < f_5(w(S_j, i+1))$. On the other hand for every selection set S we have $f_5(w(S, i)) = f_5(w(S, i+1))$. Hence f_5 does not satisfy the Pumping Lemma above.

▶ **Example 20.** The function f_5 in Example 5 is essentially the function f_2 from Example 2 applied to the subwords between the symbols #, where the outputs are aggregated with +. In a similar way one can define a min-plus automaton recognizing $f_6(w) = \sum_i f_4(w_i)$ for any $w \in \Sigma^*$ of the form $w_0\#w_1\# \ldots \#w_n$ with $w_i \in \{a, b\}^*$, where f_4 is the function computing the minimal block of b's from Example 4. We show that f_6 is not definable by PA-WA over $\mathbb{N}_{\min,+}$. Consider the following n-pumping representation: $w = (\underline{b^N} \cdot a \cdot \underline{b^N}\#)^m$ where $m \geq \varphi(2)$ (here $\max_i(|S_i|) = 2$). We index each pumping fragment with a pair (s, j) like in Example 19 and we define the subsets $S_1 \ldots S_m$ as follows: $S_j = \{(1, j), (2, j)\}$. Clearly for all j we have $f_6(w(S_j, i)) < f_6(w(S_j, i+1))$. On the other hand for every selection set S we have $f_6(w(S, i)) = f_6(w(S, i+1))$.

Consider the set of matrices $\mathbb{N}_{\min,+}^{Q \times Q}$ over the min-plus semiring. Recall that here $\oplus = \min$, $\odot = +$, $\not< = \infty$, $\not\leq = 0$, and the product of matrices $M, N \in \mathbb{N}_{\min,+}^{Q \times Q}$ is defined by $M \cdot N(p, q) = \min_r(M(p, r) + N(r, q))$. Also, recall that for any $M \in \mathbb{N}_{\min,+}^{Q \times Q}$ we denote by $\bar{M}$ the homomorphic image of M into the finite monoid $\mathbb{B}^{Q \times Q}$ (see Section 2.2). Similar as in Section 3 and Section 4, we say that $D \in \mathbb{N}_{\min,+}^{Q \times Q}$ is an idempotent if $\bar{D}$ is an idempotent in the finite monoid $\mathbb{B}^{Q \times Q}$.

The following lemma is a special property of polynomially-ambiguous automata that we exploit in the proof of Theorem 18. The proof is omitted here due to lack of space.

▶ **Lemma 21.** *Let $\mathcal{A} = (Q, \Sigma, \{M_a\}_{a\in\Sigma}, I, F)$ be a polynomially-ambiguous weighted automaton over the min-plus semiring. For every idempotent $D \in \{M_w \mid w \in \Sigma^*\}$ and for every $p, q \in Q$, there exist constants $c, d \in \mathbb{N}_{\min,+}$ and $b \in \mathbb{N}$ such that $D^{b+i}(p,q) = c \cdot i + d$ for all $i \geq 0$.*

Proof of Theorem 18. Consider a polynomially-ambiguous WA $\mathcal{A} = (Q, \Sigma, \{M_a\}_{a\in\Sigma}, I, F)$ over $\mathbb{N}_{\min,+}$ such that $f = [\![\mathcal{A}]\!]$. We take as N the constant from Lemma 6 for the finite monoid $\mathbb{B}^{Q\times Q}$. The function $\varphi : \mathbb{N} \to \mathbb{N}$ will be determined later in the proof. Consider an n-pumping representation w like in the statement of the lemma. Recall that the output for the word w is defined as $I \cdot M_w \cdot F$. By Lemma 6, for every v_k there exists a factorization $v_k = x_k y_k z_k$ such that M_{y_k} is an idempotent and $|y_k| \leq N$. We denote $D_k = M_{y_k}$ and define:

$$w \;=\; u'_0 \cdot \underline{y_1} \cdot u'_1 \cdot \underline{y_2} \cdot \ldots u'_{n-1} \cdot \underline{y_n} \cdot u'_n$$

such that each word y_k is the infix of v_k corresponding to the idempotent D_k. For the rest of the proof we denote $w_{\leq k} = u'_0 \cdot y_1 \cdot \ldots u'_{k-1}$. For every $S \subseteq \{1 \ldots n\}$ we denote by $w_{\leq k}(S, i)$ the word $w_{\leq k}$ with all y_j pumped i times for all $j < k$ such that $j \in S$.

Recall that $\mathrm{Run}_{\mathcal{A}}(w)$ is the set of all accepting runs and let $\rho \in \mathrm{Run}_{\mathcal{A}}(w)$. Every run induces two states for each $1 \leq k \leq n$: states preceding and following each word y_k. In the rest of the proof these will be the most important parts of a run. To work with them, we define the abstraction of ρ, denoted by $\bar{\rho} : \{1 \ldots n\} \to Q \times Q$, such that $\bar{\rho}(k) = (p, q)$ where p and q are the $|w_{\leq k}|$-th and $|w_{\leq k} \cdot y_k|$-th states of ρ, respectively. Similarly, for $S \subseteq \{1 \ldots n\}$, $i \geq 1$, and $\rho \in \mathrm{Run}_{\mathcal{A}}(w(S, i))$ we define $\bar{\rho} : \{1 \ldots n\} \to Q \times Q$ such that $\bar{\rho}(k) = (p, q)$ where p and q are the $|w_{\leq k}(S, i)|$-th and $|w_{\leq k}(S, i) \cdot y_k(S, i)|$-th states of ρ, respectively. We denote by $\overline{\mathrm{Run}_{\mathcal{A}}}(w)$ the set of all abstraction of runs in $\mathrm{Run}_{\mathcal{A}}(w)$. Observe that since all D_k are idempotents, $\overline{\mathrm{Run}_{\mathcal{A}}}(w(S, i)) = \overline{\mathrm{Run}_{\mathcal{A}}}(w)$ for all subsets S and $i \geq 1$.

The next step is to prove that there exists a polynomial function $p(x)$, depending only on $\mathcal{A}$, such that $|\overline{\mathrm{Run}_{\mathcal{A}}}(w)| \leq p(n)$. Let w' be the word obtained from w were each u'_i is replaced with a word u''_i of length at most $|\mathbb{B}^{Q\times Q}|$ such that $\overline{M_{u'_i}} = \overline{M_{u''_i}}$ (it is straightforward to prove that u''_i exists by pigeonhole principle). Then $|\mathrm{Run}_{\mathcal{A}}(w')| \geq |\overline{\mathrm{Run}_{\mathcal{A}}}(w)|$. Recall that $|y_i| \leq N$ and that N depends only on $|\mathbb{B}^{Q\times Q}|$. Then by definition $|w'| \leq (N + |\mathbb{B}^{Q\times Q}|) \cdot (n+1)$ and thus $|\mathrm{Run}_{\mathcal{A}}(w')| \leq r((N + |\mathbb{B}^{Q\times Q}|) \cdot (n+1))$, where r is the polynomial bounding the number of runs in $\mathcal{A}$. The claim follows for $p(n) = r((N + |\mathbb{B}^{Q\times Q}|) \cdot (n+1))$.

Fix a nonempty set $S \subseteq \{1, \ldots, n\}$ and $\rho \in \mathrm{Run}_{\mathcal{A}}(w)$. For every $k \in S$ let $b^k_{\bar{\rho}(k)}$, $c^k_{\bar{\rho}(k)}$ and $d^k_{\bar{\rho}(k)}$ be the constants from Lemma 21 such that $D_k^{b^k_{\bar{\rho}(k)}+i}[\bar{\rho}(k)] = c^k_{\bar{\rho}(k)} \cdot i + d^k_{\bar{\rho}(k)}$ for i sufficiently large. Since ρ is accepting then $c^k_{\bar{\rho}(k)}, d^k_{\bar{\rho}(k)} < +\infty$. We show that:

1. $[\![\mathcal{A}]\!](w(S, i)) = [\![\mathcal{A}]\!](w(S, i+1))$ for i sufficiently large iff there exists a run $\rho \in \mathrm{Run}_{\mathcal{A}}(w)$ such that $c^k_{\bar{\rho}(k)} = 0$ for every $k \in S$;
2. $[\![\mathcal{A}]\!](w(S, i)) < [\![\mathcal{A}]\!](w(S, i+1))$ for i sufficiently large iff for every run $\rho \in \mathrm{Run}_{\mathcal{A}}(w)$ there exists k such that $c^k_{\bar{\rho}(k)} > 0$.

Let $\rho \in \mathrm{Run}_{\mathcal{A}}(w(S, i+1))$ be a run realizing the minimum value for $i \geq i_0$. Given that D_k are idempotents one can always find a run $\rho' \in \mathrm{Run}_{\mathcal{A}}(w(S, i))$ such that $\bar{\rho}' = \bar{\rho}$ by removing one part on each y_k. In particular $|\rho'| \leq |\rho|$, which proves $[\![\mathcal{A}]\!](w(S, i)) \leq [\![\mathcal{A}]\!](w(S, i+1))$. It follows that if we prove (1) then (2) also holds. To prove (1) suppose first $[\![\mathcal{A}]\!](w(S, i)) = [\![\mathcal{A}]\!](w(S, i+1))$ for i sufficiently large. Let $\rho \in \mathcal{A}(w(S, i+1))$ and $\rho' \in \mathcal{A}(w(S, i))$ be the previous runs realizing the minimum and its shortening, respectively.

Since $|y_k| \leq N$ we assume a universal bound i_0 such that $b^k_{\bar{\rho}(k)} = i_0$ for all k in Lemma 21. By Lemma 21 $D_k^{i_0+i+1}[\bar{\rho}(k)] = c^k_{\bar{\rho}(k)} \cdot (i+1) + d^k_{\bar{\rho}(k)}$. If $c^k_{\bar{\rho}(k)} > 0$ for some k then the inequality $[\![\mathcal{A}]\!](w(S, i_0 + i)) \leq [\![\mathcal{A}]\!](w(S, i_0 + i + 1))$ would be sharp, which is a contradiction. For the other direction suppose there exists a run $\rho \in \mathrm{Run}_{\mathcal{A}}(w)$ such that $c^k_{\bar{\rho}(k)} = 0$ for every $k \in S$. Then for every $i \geq 0$ there exists a run $\rho_i \in \mathrm{Run}_{\mathcal{A}}(w(S, i_0 + i))$ such that $|\rho_i| \leq |\rho| + \sum_k d^k_{\bar{\rho}(k)}$. Since $[\![\mathcal{A}]\!](w(S, i_0 + i)) \leq [\![\mathcal{A}]\!](w(S, i_0 + i + 1)) \leq |\rho| + \sum_k d^k_{\bar{\rho}(k)}$ it follows that $[\![\mathcal{A}]\!](w(S, i_0 + i)) = [\![\mathcal{A}]\!](w(S, i_0 + i + 1))$ for i sufficiently large.

Given the previous discussion, let $\bar{R}_k = \{\bar{\rho} \in \overline{\mathrm{Run}}_{\mathcal{A}}(w) \mid c^k_{\bar{\rho}(k)} > 0\}$ for every $k \in \{1, \ldots, n\}$. The set $\bar{R}_k$ represents indirectly the runs that will grow when pumping $w(\{k\}, i)$. Then, we can restate (2) as: $[\![\mathcal{A}]\!](w(S, i)) < [\![\mathcal{A}]\!](w(S, i+1))$ for i sufficiently large iff $\bigcup_{k \in S} \bar{R}_k = \overline{\mathrm{Run}}_{\mathcal{A}}(w)$.

We are ready to prove the theorem. Fix a partition $S_1, \ldots, S_m$ for some $m \geq \varphi(\max |S_l|)$. Suppose the first condition is not true, namely, for all j there exists arbitrarily big values i such that $f(w(S_j, i)) \neq f(w(S_j, (i+1)))$. From (2) it follows that $f(w(S_j, i)) < f(w(S_j, i+1))$ for i sufficiently large and $\bigcup_{k \in S_j} \bar{R}_k = \overline{\mathrm{Run}}_{\mathcal{A}}(w)$ for every $j \leq m$. Let $L = \max |S_l|$. We assume that $L > 1$, otherwise every selection S contains a whole set S_k for some k and we are done by (2). To construct the set $S = \{k_1, \ldots, k_m\}$ we define by induction the sets G_j. Let $G_0 = \overline{\mathrm{Run}}_{\mathcal{A}}(w)$ and for every $j \in \{1, \ldots, m\}$ let $G_j = \overline{\mathrm{Run}}_{\mathcal{A}}(w) \setminus \bigcup_{l \leq j} \bar{R}_{k_l}$. Intuitively, G_j correspond to runs that are not covered by the set $\{k_1, \ldots, k_j\}$. For the inductive case, suppose that $G_j \neq \varnothing$. Since $\bigcup_{k \in S_{j+1}} \bar{R}_k = \overline{\mathrm{Run}}_{\mathcal{A}}(w)$, by the pigeonhole principle there exist $k_{j+1} \in S_{j+1}$ such that $|\bar{R}_{k_{j+1}} \cap G_j| \geq |G_j|/|S_{j+1}|$. We add k_{j+1} to S and so $|G_{j+1}| \leq |G_j| - |G_j|/|S_{j+1}| = |G_j| \cdot (|S_{j+1}| - 1)/|S_{j+1}| \leq |G_j| \cdot (L-1)/L$. Suppose this procedure continues until $j = m$ and $G_m \neq \varnothing$. Then $1 \leq |\overline{\mathrm{Run}}_{\mathcal{A}}(w))| \cdot ((L-1)/L)^m$, and $|\overline{\mathrm{Run}}_{\mathcal{A}}(w))| \geq (L/(L-1))^m$. However, we know that $|\overline{\mathrm{Run}}_{\mathcal{A}}(w))|$ is bounded by a polynomial function $p(n)$ depending on $|\mathcal{A}|$. Thus, it suffices to choose φ such that $m \geq \varphi(L)$ implies $(L/(L-1))^m > p(L \cdot m) \geq p(n) \geq |\overline{\mathrm{Run}}_{\mathcal{A}}(w))|$ (recall that $S_1, \ldots, S_m$ is a partition of $\{1, \ldots, n\}$ and $L \cdot m \geq n$). Therefore, $G_m = \varnothing$ and thus $\bigcup_{k \in S} \bar{R}_k = \overline{\mathrm{Run}}_{\mathcal{A}}(w)$, which concludes the proof. ◀

6 Conclusions

We have shown three pumping lemmas for three different classes of functions. We believe that the last pumping lemma in Section 5 could be proved for a wider class of functions that would contain the class $\mathbb{N}_{+,\times}$, but this is left for future work. As a corollary of our results, we showed that regular functions over $\mathbb{N}_{\min,+}$ form a strict hierarchy, namely:

$$\text{U-WA} \subsetneq \text{FA-WA} \subsetneq \text{PA-WA} \subsetneq \text{WA}.$$

All strict inclusions, except for PA-WA $\subsetneq$ WA, could be extracted from the analysis of examples in [16]. However, our results provide a general machinery to prove such results.

References

1 Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? In Tevfik Bultan and Pao-Ann Hsiung, editors, *Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings*, volume 6996 of *Lecture Notes in Computer Science*, pages 482–491. Springer, 2011. doi:10.1007/978-3-642-24372-1_37.

2 Rajeev Alur, Loris D'Antoni, Jyotirmoy Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *LICS*, pages 13–22. IEEE Computer Society, 2013.

3 Rajeev Alur and Mukund Raghothaman. Decision problems for additive regular functions. In *Automata, Languages, and Programming*, pages 37–48. Springer, 2013.

4 Marcelo Arenas, Martin Munoz, and Cristian Riveros. Descriptive complexity for counting complexity classes. In *Logic in Computer Science (LICS), 2017 32nd Annual ACM/IEEE Symposium on*, pages 1–12. IEEE, 2017.

5 Udi Boker, Krishnendu Chatterjee, Thomas A Henzinger, and Orna Kupferman. Temporal specifications with accumulative values. *ACM Transactions on Computational Logic (TOCL)*, 15(4):27, 2014.

6 Thomas Colcombet, Denis Kuperberg, Amaldev Manuel, and Szymon Toruńczyk. Cost functions definable by min/max automata. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 29:1–29:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.29.

7 Karel Culik II and Jarkko Kari. Image compression using weighted finite automata. In *Mathematical Foundations of Computer Science 1993*, pages 392–402. Springer, 1993.

8 Laure Daviaud, Pierre-Alain Reynier, and Jean-Marc Talbot. A generalised twinning property for minimisation of cost register automata. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 857–866. ACM, 2016. doi:10.1145/2933575.2934549.

9 Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theor. Comput. Sci.*, 380(1-2):69–86, 2007.

10 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer, 1st edition, 2009.

11 Andrzej Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fund. Math*, 49(129-141):13, 1961.

12 Ronald Fraïsé. Sur quelques classification des systemes de relations. *Université d\'Alger, Publications Scientifiques, Série A*, 1:35–182, 1984.

13 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

14 Daniel Kirsten. A burnside approach to the termination of mohri's algorithm for polynomially ambiguous min-plus-automata. *ITA*, 42(3):553–581, 2008. doi:10.1051/ita:2008017.

15 Daniel Kirsten and Sylvain Lombardy. Deciding unambiguity and sequentiality of polynomially ambiguous min-plus automata. In *STACS*, pages 589–600, 2009.

16 Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theor. Comput. Sci.*, 327(3):349–373, 2004.

17 Stephan Kreutzer and Cristian Riveros. Quantitative monadic second-order logic. In *LICS*, pages 113–122. IEEE Computer Society, 2013.

18 Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. In Werner Kuich, editor, *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings*, volume 623 of *Lecture Notes in Computer Science*, pages 101–112. Springer, 1992. doi:10.1007/3-540-55719-9_67.

19 Leonid Libkin. *Elements of finite model theory*. Springer Science & Business Media, 2013.

20 Filip Mazowiecki and Cristian Riveros. Maximal partition logic: Towards a logical characterization of copyless cost register automata. In *24th EACSL Annual Conference on*

Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany, pages 144–159, 2015.

21 Filip Mazowiecki and Cristian Riveros. Copyless cost-register automata: Structure, expressiveness, and closure properties. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 53:1–53:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.53.

22 Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

23 Joël Ouaknine and James Worrell. On the positivity problem for simple linear recurrence sequences,. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 318–329. Springer, 2014. doi:10.1007/978-3-662-43951-7_27.

24 Joël Ouaknine and James Worrell. Ultimate positivity is decidable for simple linear recurrence sequences. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 330–341. Springer, 2014. doi:10.1007/978-3-662-43951-7_28.

25 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1993.

26 Jean-Éric Pin. Mathematical foundations of automata theory. *Lecture notes LIAFA, Université Paris*, 7, 2010.

27 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009. URL: http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521844253.

28 M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.

29 Andreas Weber. Finite-valued distance automata. *Theor. Comput. Sci.*, 134(1):225–251, 1994.

Closure of Resource-Bounded Randomness Notions Under Polynomial-Time Permutations

André Nies
Department of Computer Science, The University of Auckland,
Private Bag 92019, Auckland, New Zealand
andre@cs.auckland.ac.nz

Frank Stephan
Department of Mathematics and Department of Computer Science,
National University of Singapore,
10 Lower Kent Ridge Road, Block S17, Singapore 119076, Republic of Singapore
fstephan@comp.nus.edu.sg

Abstract

An infinite bit sequence is called recursively random if no computable strategy betting along the sequence has unbounded capital. It is well-known that the property of recursive randomness is closed under computable permutations. We investigate analogous statements for randomness notions defined by betting strategies that are computable within resource bounds. Suppose that S is a polynomial time computable permutation of the set of strings over the unary alphabet (identified with $\mathbb{N}$). If the inverse of S is not polynomially bounded, it is not hard to build a polynomial time random bit sequence Z such that $Z \circ S$ is not polynomial time random. So one should only consider permutations S satisfying the extra condition that the inverse is polynomially bounded. Now the closure depends on additional assumptions in complexity theory.

Our first main result, Theorem 4, shows that if BPP contains a superpolynomial deterministic time class, such as $\mathsf{DTIME}(n^{\log n})$, then polynomial time randomness is not preserved by some permutation S such that in fact both S and its inverse are in P. Our second main result, Theorem 11, shows that polynomial space randomness is preserved by polynomial time permutations with polynomially bounded inverse, so if $\mathsf{P} = \mathsf{PSPACE}$ then polynomial time randomness is preserved.

2012 ACM Subject Classification Theory of computation → Complexity classes, Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Computational Complexity, Randomness via Resource-bounded Betting Strategies, Martingales, Closure under Permutations

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.51

Funding A. Nies is supported in part by the Marsden Fund of the Royal Society of New Zealand, UoA 13-184. F. Stephan is supported in part by the Singapore Ministry of Education Academic Research Fund Tier 2 grant MOE2016-T2-1-019 / R146-000-234-112. Part of this work was done while F. Stephan was on sabbatical leave at the University of Auckland. The work was completed while Nies visited the Institute for Mathematical Sciences at NUS during the 2017 programme "Aspects of Computation".

Acknowledgements The authors would like to thank Eric Allender, Klaus Ambos-Spies, Alexander Galicki, Elvira Majordomo, and Wolfgang Merkle for discussions and comments.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 51; pp. 51:1–51:10

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Formal randomness notions for infinite bit sequences can be studied via algorithmic tests. A hierarchy of such notions has been introduced. See e.g. Downey and Hirschfeldt [4] or Nies [11, Ch. 3] for definitions and basic properties, and also Li and Vitányi [8]. Criteria for good randomness notions include robustness under certain computable operations on the bit sequences. In the simplest case, such an operation is a computable permutation of the bits. For a permutation S of $\mathbb{N}$ and an infinite bit sequence Z, identified with a subset of $\mathbb{N}$, by $Z \circ S$ we denote the sequence Y such that $Y(n) = Z(S(n))$. (Note that, when viewed as a subset of $\mathbb{N}$, $Z \circ S$ equals $S^{-1}(Z)$.) We say that a class $\mathcal{C}$ of bit sequences is *closed under all members of a class $\mathcal{G}$ of permutations* if $Z \in \mathcal{C}$ implies $Z \circ S \in \mathcal{C}$ for each $S \in \mathcal{G}$.

A central notion of randomness was introduced by Martin-Löf [9]. A Martin-Löf test is a uniformly Σ^0_1 sequence $\langle G_m \rangle_{m\in\mathbb{N}}$ such that the uniform measure of G_m is at most 2^{-m}. Z fails such a test if $Z \in \bigcap_m G_m$; otherwise Z passes the test. Z is Martin-Löf random if it passes each such test. Clearly this randomness notion is closed under computable permutations S: if $Z \circ S$ fails a Martin-Löf-test $\langle G_m \rangle_{m\in\mathbb{N}}$, then Z fails the test $\langle S^{-1}(G_m) \rangle_{m\in\mathbb{N}}$. The weaker notion of Schnorr randomness [13], where one also requires that the measure of G_m is a computable real uniformly in m, is closed under computable permutations by a similar argument. Recursive randomness [13] (see e.g. [11, Ch. 7] as a recent reference) is defined via failure of all computable betting strategies (martingales), rather than by a variant of Martin-Löf's test notion. Nonetheless, by a more involved argument, implicit in [3, Section 4.1], it is closed under computable permutations. Also see Nies [11, Thm. 7.6.24] and Kjos-Hanssen, Nguyen and Rute [7].

Our main purpose is to study analogues of this result in computational complexity theory. In order to guarantee compatibility with the theory developed in Downey and Hirschfeldt [4] and Nies [11] we view sets of numbers (i.e., infinite bit sequences), rather than sets of strings over an alphabet of size at least 2, as our principal objects of study. We note that work of Lutz, Mayordomo, Ambos-Spies and others, beginning in the 1980s and surveyed in Ambos-Spies and Mayordomo [1], studied sets of strings: martingales bet on the strings in length-lexicographical order. Such languages can be identified with bit sequences via this order of strings, but the time bounds imposed on martingales are exponentially larger when they bet on strings.

To be able to apply the notions of resource bounded computability to infinite bit sequences and permutations, we will identify such bit sequences with subsets of the set $\{0\}^*$ of unary strings (also called tally languages). We view permutations as acting on $\{0\}^*$. A bit sequence is *polynomial time random* if no polynomial time computable bettings strategy succeeds on the sequence. This notion was briefly introduced by Schnorr [13], studied implicitly in the above-mentioned work of Lutz, Mayordomo, Ambos-Spies and others, and in more explicit form in Yongge Wang's 1996 thesis [15].

Our leading question is: *under which polynomial time computable permutations S is polynomial time randomness closed?* If S^{-1} is not polynomially bounded, we build a polynomial time random bit sequence Z such that $Z \circ S$ is not polynomial time random. After that, we will assume that S satisfies the extra condition that its inverse is polynomially bounded. Now the closure depends on additional assumptions in complexity theory:

- The first result, Theorem 4, shows that if BPP contains a superpolynomial deterministic time class, such as $\mathsf{DTIME}(n^{\log n})$, then polynomial time randomness is not preserved by some permutation S such that both S and its inverse are in P.
- The second result, Theorem 11, shows that PSPACE-randomness is preserved by polynomial time permutations with polynomially bounded inverse; so if $\mathsf{P} = \mathsf{PSPACE}$ then polynomial time randomness is preserved by such permutations.

Broadly speaking, the idea for the first result, Theorem 4, is as follows. Choose an $O(n^{\log n})$ time computable martingale M only betting on odd positions $1, 3, 5, \ldots$ that dominates (up to a positive factor) all polynomial time computable martingales that only bets on odd positions. Use the hypothesis in order to take a language $A \in \mathsf{BPP}$ which tells at which extension of a string of odd length M does not increase. Now let B be a highly random set (albeit B can be chosen in E). Let Z be the bit sequence that copies $B(n)$ at position $2n$, and takes the value of A at the string $Z(0) \ldots Z(2n)$ at position $2n + 1$. Then one can verify that Z is polynomial time random. If $\widehat{Z}$ is a rearrangement of the bits of Z so that a sufficiently large block of bits of B is interspersed between bits determined by A, then we can use these bits of B as random bits required in a randomised polynomial time algorithm for A. This will show that $\widehat{Z}$ is not polynomial time random.

The second result, Theorem 11, closely follows Buhrman, van Melkebeek, Regan, Sivakumar and Strauss [3, Section 4.1], which introduces and studies resource-bounded betting games. It actually shows that PSPACE-randomness is closed under certain polynomial time scanning functions, which, unlike permutations, can uncover the bits of a set in an order determined by previous bits. Each permutation in question can be seen as a scanning function of the appropriate kind. (We note that removing the resource bounds from Theorem 11 yields a proof that recursive randomness is closed under computable permutations, and in fact under computable scanning functions that scan each position.) Thm. 5.6 in Buhrman et al. [3] is a related result based on the same methods developed there; however, in that result an assumption on the existence of certain pseudorandom generators is made, while our Theorem 11 does not rest on any unproven assumptions.

We note another notion of robustness for randomness notions. One can easily adapt all the randomness notions to an alphabet other than $\{0, 1\}$. Base invariance says that the randomness notion is preserved when one replaces a sequence over one alphabet by a sequence over a different alphabet that denotes the same real number. Brattka, Miller and Nies [2] have shown this for recursive randomness, and Figueira and Nies [5] have shown it for polynomial time randomness, each time relying on the connection of randomness of a real with differentiability at the real of certain effective functions.

Using Figueira and Nies [5], Nies [12] provides a characterisation of polynomial time randomness for real numbers in terms of differentiability of all polynomial time computable nondecreasing functions on the reals.

2 Preliminaries

For a bound h, as usual $\mathsf{DTIME}(h)$ denotes the languages A computable in time $O(h)$. Informally we often say that A is computable in time h. As in Ambos-Spies and Mayordomo [1], we require that martingales have rational values.

▶ **Definition 1.** A martingale M is a function from $\{0,1\}^*$ to $\{q \in \mathbb{Q} : q > 0\}$ satisfying $M(x) = (M(x0) + M(x1))/2$ for all $x \in \{0,1\}^*$. A martingale succeeds on a set Z if $\limsup_n M(Z \upharpoonright n) = \infty$. One says that a martingale *does not bet at a position* n if $M(x0) = M(x1)$ for each $x \in \{0,1\}^n$.

One says that Z is recursively random if no computable martingale succeeds on Z.

Each polynomial in this paper will be non-constant and have natural number coefficients. For a polynomial time version of recursive randomness, we have to be careful how to define polynomial time computability for a martingale: as in [3], a positive rational number q is presented by a pair $\langle k, n\rangle$ consisting of a denominator and a numerator (both written in

binary) such that $q = k/n$ in lowest terms. A martingale M is polynomial time computable if on input x one can determine $M(x)$ in this format in polynomial time. Z is polynomial time random if no such martingale succeeds on Z. In a similar way one defines exponential time randomness.

A martingale M is polynomial space computable if $M(x)$ can be computed in polynomial space (including the space needed to write the output). Z is polynomial space random if no such martingale succeeds on Z.

We first show that polynomial time randomness fails to be closed under polynomial time computable permutations S that are "dishonest" in the sense that $S(n)$ can be much less than n.

▶ **Theorem 2.** *Let S be a polynomial time permutation of $\{0\}^*$ such that for each polynomial p, there are infinitely many n with $p(S(n)) \leq n$. There is a polynomial time random Z computable in time $2^{O(n)}$ such that $Z \circ S$ is not polynomial time random.*

Clearly a permutation S as in Theorem 2 exists: Let $(p_k)_{k\in\mathbb{N}}$ list the non-constant polynomials with natural coefficients in such a way that for $u \leq n$, $O(n^2)$ steps suffice to verify whether $p_k(u) \leq n$. On input n of the form $\langle k, i\rangle$, see whether $p_k(\langle k, 0\rangle) \leq n$. If not let $S(n) = \langle k, i+1\rangle$. If so and n is least such, let $S(n) = \langle k, 0\rangle$. Otherwise $S(n) = n$.

Proof of Theorem 2. Nies [11, Section 7.4] provided a construction template for recursively random sets, going back to Schnorr's work. We adapt some parts of this template to the resource bounded setting.

Let $\langle B_k\rangle$ be an effective listing of the polynomial time martingales with positive rational values. We may assume that B_k is computable in time $p_k(n) = k(n^k + 1)$.

For each n, let $B_{k,n}$ be the martingale with initial capital 1 that does not bet until its input reaches length n, and then uses the same betting factors as B_k. Thus,

$$B_{k,n}(x) = \frac{B_k(x)}{B_k(x \upharpoonright n)}$$

for any string x of length at least n. Let $\widetilde{p}_{k,n}$ be a polynomial so that $B_{k,n}(x)$ for $|x| \geq n$ can be computed in time $\widetilde{p}_{k,n}(|x|)$.

We inductively define a sequence of numbers. Let $n_0 = 0$, and let n_{k+1} be the least $n > n_k$ such that $q_k(S(n)+1) \leq n$, where q_k is a polynomial time bound for the martingale $\sum_{r\leq k} 2^{-r} B_{r,n_r}$ and $q_k(n) \geq n + 2$. Let $L = \sum_r 2^{-r} B_{r,n_r}$. Note that L is a rational-valued martingale, because on inputs of length at most n_k, all the B_{r,n_r} for $r > k$ together contribute 2^{-k}.

Let now Z be the left-most non-ascending path of L: $Z(m) = 0$ if $L(Z \upharpoonright m\hat{}0) \leq L(Z \upharpoonright m)$, and $Z(m) = 1$ otherwise. Since L does not succeed on Z and L multiplicatively dominates each B_k, the set Z is polynomial time random.

Note that since $S \in \mathsf{P}$, from n we can in polynomial time recursively recover the sequence $n_0, q_0, n_1, q_1, \ldots$ and thereby compute the maximal k such that $n_k < n$. In particular we can decide whether n is of the form n_{k+1} for some k. By definition, for $n = n_{k+1}$ we have $q_k(S(n)+1) \leq n$ and hence $S(n) + 1 < n_{k+1}$. Since q_k as a time bound is sufficient to determine $L(y)$ for strings y of length $S(n) + 1$, the bit $Z \circ S(n)$ can be computed in time polynomial in n. Hence $Z \circ S$ is not polynomially random.

We can ensure such a set Z is computable in time $2^{O(n)}$ by choosing the listing $\langle B_k\rangle$ appropriately. ◀

▶ Remark. We note that methods involving the $\langle B_{k,n} \rangle$ similar to the above can be used to show that each class $\mathsf{DTIME}(h)$ with superpolynomial time constructible h contains a polynomial time random (tally) set. We have to initiate a copy $\langle B_{k,n} \rangle$ of B_k finitely many times until a length n is reached such that for $m \geq n$, $h(m)$ time is sufficient to simulate its behaviour on strings of length m.

3 If BPP Contains a Superpolynomial Time Class Then Closure Fails

▶ **Definition 3.** A permutation S of $\{0\}^*$ is called *fully polynomial time computable* if both S and S^{-1} are polynomial time computable.

A complexity theoretic assumption considerably weaker than $\mathsf{BPP} = \mathsf{EXP}$ suffices for non-closure.

▶ **Theorem 4.** *Suppose that* $\mathsf{DTIME}(h) \subseteq \mathsf{BPP}$ *for some time constructible function* h *that dominates all the polynomials. Then there are a polynomial time random set* $Z \in \mathsf{DTIME}(2^{3n})$ *and a fully polynomial time computable permutation* S *such that* $Z \circ S$ *is not polynomial time random.*

Proof. We may assume that $h(n) \leq n^{\log n}$. It is well-known that whenever a martingale in a certain complexity class succeeds on a set Z then there is also a successful martingale in the same class betting only on even positions, or there is a successful martingale betting only on the odd positions.

The construction has two steps. Firstly, by standard methods discussed at the end of Section 2, one can build a martingale M in $\mathsf{DTIME}(h)$ which bets only on odd positions, and dominates up to a multiplicative constant all polynomial time martingales betting on odd positions. Let

$$A = \{x \in \{0,1\}^* : x \text{ has odd length and } M(x1) < M(x0)\}.$$

The set A is in $\mathsf{DTIME}(h)$ and hence by assumption in BPP.

Secondly, let $B \subseteq \{0\}^*$ be a language on which no martingale in $\mathsf{DTIME}(2^{4 \cdot n})$ succeeds. Again by standard methods one can ensure that B is in $\mathsf{DTIME}(2^{5 \cdot n})$. Define a set $Z \subseteq \mathbb{N}$ as follows:

$$Z(2n) = B(n); \; Z(2n+1) = A(Z \upharpoonright 2n+1).$$

We may visualise Z as follows:

B	A	B	A	B	A	B	A	B	A	B	A	B	...
$B(0)$	$A(Z \upharpoonright 1)$	$B(1)$	$A(Z \upharpoonright 3)$	$B(2)$	$A(Z \upharpoonright 5)$	...							

Clearly $Z \in \mathsf{DTIME}(2^{3n})$. It is claimed that Z is polynomial time random. As the martingale M only bets on odd positions, Z is defined such that M never gains capital on Z. As M is universal among the martingales computable in polynomial time with this property, no martingale betting on the odd positions succeeds on Z.

Suppose now that L is a polynomial time martingale which bets on the even positions and note that one can compute in time $O(h(n))$ from $B(0), B(1), \ldots, B(n)$ inductively the values $Z(0), Z(1), \ldots, Z(2n+1)$, as for every x of length $2n+1$ the value $A(x)$ can be computed in time $h(n)$. Thus if L succeeds on Z then there is a new martingale N succeeding on B which satisfies that

$$N(B \upharpoonright n+1) = L(Z \upharpoonright 2n)$$

and which uses that $Z(2n) = B(n)$ while the bits of Z at odd positions on which L does not bet can be computed as indicated above from the other bits. To compute $N(x)$ for x of length $2n$ takes $q(n)+\sum_{i<n} h(2i+1)$ steps for some polynomial q. So $N \in \mathsf{DTIME}(n^{O(\log n)})$, which contradicts the assumption that no such martingale computable in time $O(2^{4n})$ succeeds on B. This verifies the claim.

Since $A \in \mathsf{BPP}$, there is a polynomial p such that an appropriate randomised algorithm $\mathcal{R}$ on input $x \in \{0,1\}^{2n+1}$ computes $A(x)$ in time $p(n)$, with error probability 2^{-4n-2}, using $p(n)$ random bits. Now consider the sequence $\widehat{Z}$ consisting for $n = 0, 1, \ldots$ of $p(n)$ bits taken from B followed by the bit $Z(2n+1)$. Again we visualise $\widehat{Z}$:

B	A	B	B	B	A	B	B	B	B	B	B	A	...
$p(0)$			$p(1)$						$p(2)$				

Formally one can define $\widehat{Z}$ from Z as follows:

$$\begin{aligned} &\text{for } m < p(n), \\ &\widehat{Z}((\sum_{k<n} p(k)) + n + m) \;=\; B((\sum_{k<n} p(k)) + m) \;=\; Z(2(\sum_{k<n} p(k) + m)); \\ &\widehat{Z}((\sum_{k\le n} p(k)) + n) \;=\; Z(2n+1) = A(Z \upharpoonright 2n+1). \end{aligned}$$

This mapping is given by a permutation S so that $\widehat{Z}(r) = Z(S(r))$ for all positions r. So if $r = (\sum_{k<n} p(k)) + n + m$ then $S(r) = 2(\sum_{k<n} p(k)) + 2m$ and if $r = (\sum_{k\le n} p(k)) + n$ then $S(r) = 2n+1$, for all m, n with $m < p(n)$. The permutation S and its inverse satisfy that the mappings $0^k \mapsto 0^{S(k)}$ and $0^k \mapsto 0^{S^{-1}(k)}$ on the unary strings $\{0\}^*$ are polynomial time computable, thus the S is of the form as required; to see this note that for a polynomial p also the mapping $n \mapsto \sum_{k<n} p(k)$ is a polynomial; similarly for a function bounded by a polynomial.

Now it will be shown that $\widehat{Z}$ is not polynomial time random. Note that there are 2^{2n+1} strings of length $2n+1$. Given a string of $p(n)$ random bits, the probability that when using these bits the randomised algorithm $\mathcal{R}$ computes $A(x)$ correctly for all $x \in \{0,1\}^{2n+1}$ is at least $1 - 2^{2n+1} \cdot 2^{-4n-2} = 1 - 2^{-2n-1}$. We want to show that B provides random bits that allow $\mathcal{R}$ to correctly compute A for almost all inputs. Otherwise, we can build a martingale M computable in time $2^{10 \cdot n}$ which succeeds on B: The martingale M splits its capital into bins of value 2^{-n-1} and for each block of $p(n)$ bits starting at $\sum_{k<n} p(k)$, it takes the value 2^{-n-1} from the corresponding bin and bets it on the strings y consisting of $p(n)$ bits that do not compute all values of $A(x)$ with $x \in \{0,1\}^{2n+1}$ correctly using $\mathcal{R}$. This condition can be checked for these bits in the time bound given as it involves running $\mathcal{R}$ with y as the random bits on all strings x of length $2n+1$ and comparing the result with $A(x)$ for all $2^{p(n)}$ choices of random bits y. After these simulations, M distributes the capital from the bin evenly on those strings of random bits which cause $\mathcal{R}$ to make an error. After having processed the bits from the block of $p(n)$ bits, the capital in this bin remains unchanged by future bets. The set of random strings y on which the computation of some of the $A(x)$ in $x \in \{0,1\}^{2n+1}$ is false has at most the probability $2^{-4n-2} \cdot 2^{2n+1} = 2^{-2n-1}$. Therefore the capital from the bin multiplies at least by 2^{n+1} during the block and reaches the value 1.

For the time bound on M, whenever the input has length between $\sum_{k<n} p(k)$ and $\sum_{k\le n} p(k)$, the martingale computes 2^{n+1} values $A(x)$ for $x \in \{0,1\}^{2n+1}$ with respect to $p(n)$ random bits taking $2^{p(n)}$ possible choices. However, for all polynomials and almost all n, $p(n) \le \sum_{k<n} p(k)$, as the degree of the sum-polynomial of p is by one above the degree

of p and the polynomial p is positive. Thus, for such n, when $n' = \sum_{k<n} p(k)$ is a lower bound on the length of the input to the martingale M then $p(n) \leq n'$ and $2n+1 \leq n'$ and thus the whole computations can be handled in time $O(2^{3n'})$.

If there are infinitely many blocks in B where the random bits of this block do not compute all $A(x)$ with x of the corresponding length correctly, then this martingale succeeds, contrary to the assumption on B. So, for almost all n, the block of $p(n)$ random bits in $\widehat{Z}$ before $A(Z \upharpoonright 2n)$ permits to compute this value correctly.

Now this property will be used to construct a polynomial time martingale H which succeeds on $\widehat{Z}$. Let $\tilde{A}(n)$ denote $A(Z \upharpoonright 2n+1)$. Given $p(n)$ random bits from B preceding $\tilde{A}(n)$ in $\widehat{Z}$, the martingale H archives these bits without betting on them. It then bets half of its capital on the value for $\tilde{A}(n)$ computed from these random bits; note that due to $\tilde{A}(0), \tilde{A}(1), \ldots, \tilde{A}(n-1)$ and $B(0), B(1), \ldots, B(n)$ being coded in $\widehat{Z}$ in positions before that of $\tilde{A}(n)$, when the bet for $\tilde{A}(n) = Z(2n+1)$ has to be made, one can retrieve besides the random bits also $Z(0)Z(1)\ldots Z(2n)$ from the history. So one can use the random bits to compute the value almost always correctly. Thus the martingale H will only finitely often place a wrong bet and lose some of its capital, but for almost all $\tilde{A}(n)$ predict the value correctly and multiply its capital by $3/2$. Thus the martingale succeeds. As all the operations above are polynomial time computable, the set $\widehat{Z}$ is not polynomial time random. ◀

The proof of Theorem 4 can be adjusted to obtain a corollary.

▶ **Corollary 5.** *Let $A, B \subseteq \{0\}^*$. Suppose that A is in* BPP *and B is* EXP*-random relative to A. Then A is polynomial time computable relative to B, and in particular not polynomial time random relative to B.*

Proof. For ease of notation, we often write $A(n)$ in place of $A(0^n)$ and so on; however, both A and B are viewed as subsets of $\{0\}^*$.

There is a polynomial time algorithm and a polynomial p such that the algorithm uses $p(n)$ random bits to compute $A(n)$ with error probability 2^{-n}. As in the theorem above, one can now query B for getting the random bits and the places where the queries are asked are different for n, m whenever $n \neq m$. So there is a polynomial q with $q(n) + p(n) = q(n+1)$ for all n and where the algorithm asks the bits of B at $q(n), q(n)+1, \ldots, q(n)+p(n)-1$ to compute $A(n)$.

If now there is an error, then an exponential time martingale relative to A can make sufficient profit, as only a slim minority of the possiblities of the bits of B from $q(n)$ to $q(n)+p(n)-1$ are realised. This contradicts the assumption that B is random relative to A. Hence A can be computed relative to B by this algorithm with only finitely many errors; these can then be corrected by a finite table holding the correct values for the positions where the algorithm makes an error. ◀

▶ Remark. In the proof of Theorem 4, $Z = \tilde{A} \oplus B$ is polynomial time random; however, $\tilde{A}$ is not polynomial time random relative to B, as the rearrangement with S shows. Note that van Lambalgen's Theorem [14] says that in a recursion-theoretic setting, $\tilde{A} \oplus B$ is random iff (a) B is random and (b) $\tilde{A}$ is random relative to B. Thus, under the assumption that BPP = EXP, one of the directions of the van Lambalgen Theorem does not hold for polynomial time randomness.

The corollary also shows that one can choose, under the assumption that BPP contains a superpolynomial time class, sets $A, B \subseteq \{0\}^*$ such that A is polynomial time random, B is polynomial time random relative to A and A is polynomial time computable relative to B. Hence this assumption implies that A is a basis for polynomial time randomness

even though A is polynomial time random itself. This contrasts with the setting of Martin-Löf randomness randomness in recursion theory: a basis for Martin-Löf randomness has to be trivial and therefore cannot be random [6, 10]. On the other hand, the bases for recursive randomness include every set below the halting problem that is not diagonally noncomputable (DNC), but no set of PA degree [6]. Every high set computes a recursively random set, and an incomplete high r.e. set is not DNC. So a recursively random set can be a basis for recursive randomness.

4 If P = PSPACE Then Closure Holds

We say that $Z \subseteq \mathbb{N}$ is *polynomial space random* if no martingale computable in polynomial space succeeds on Z. In this section we show that polynomial space randomness is closed under fully polynomial time computable permutations in the sense of Definition 3. If $\mathsf{P} = \mathsf{PSPACE}$ this closure property applies to polynomial time randomness as well.

In fact we show a stronger closure property where the permutations are generalised to certain non-monotonic scanning rules, which adaptively specify an order in which bits are read. We modify the argument given by Buhrman, van Melkebeek, Regan, Sivakumar and Strauss [3, Section 4.1], which was not concerned with polynomial space randomness, but rather was geared to the context of Lutz's theory of resource bounded measure. As already mentioned, in that theory, the positions a martingale bets on are strings in some non-unary alphabet. Such strings can be suitably encoded by natural numbers; however, the resource bounds change when one converts such a martingale into one in the sense of our Definition 1. The next two definitions formalise the idea of betting on bit positions in an order chosen adaptively by the betting strategy. We take some key technical concepts from [3, Section 4.1], somewhat changing the terminology in order to make it compatible with the one of Nies [11, Section 7.5] where non-monotonic randomness notions are studied.

▶ **Definition 6.** A *scanning function* is a function $V\colon \{0,1\}^* \to \{0\}^*$ such that $V(\alpha) \neq V(\alpha \upharpoonright i)$ for each $\alpha \in \{0,1\}^*$ and each $i < |\alpha|$. In the context of V, we will call a string α a *run of* V, thinking of α as a sequence of answers to oracle queries. We will call $V(\alpha \upharpoonright i)$ the *i-th query in the run of V on α*.

As before, subsets of $\mathbb{N}$ will be identified with languages over the unary alphabet $\{0\}$. For $Z \subseteq \mathbb{N}$ let $Z \circ V \subseteq \mathbb{N}$ be the set Y such that $Y(i) = Z(V(Y \upharpoonright i))$ for each i.

▶ **Definition 7.** A *non-monotonic betting strategy* G is a pair (V, B) such that V is a scanning function and B is a martingale. G succeeds on $Z \subseteq \{0\}^*$ if $\lim_n B(Z \circ V \upharpoonright n) = \infty$.

One says that a non-monotonic betting strategy G is computable in polynomial space if both V and B are computable in polynomial space. One says that $Z \subseteq \mathbb{N}$ is *non-monotonically polynomial space random* if no such betting strategy succeeds on Z.

Another concept we need is that of consistency between a run α of V and a string w.

▶ **Definition 8.** For bit strings α, w, we write $\alpha \sim_V w$ if for each $j < |\alpha|$, if the j-th query x in the run of V on α is less than $|w|$, then $w(x) = \alpha(j)$.

▶ **Definition 9.** For a function $g\colon \mathbb{N} \to \mathbb{N}$, one says that V is *g-filling* if for each n and each run α of length $g(n)$, we have $\forall r < n\, \exists i\, V(\alpha \upharpoonright i) = r$.

▶ **Lemma 10.** *Suppose V is g-filling. Let $|\alpha| \geq i := g(|w|)$. Then $\alpha \sim_V w$ iff $\alpha \upharpoonright i \sim_V w$.*

To see this, note that by the definition of being g-filling, any query q with $q < |w|$ has to be asked before stage $g(|w|)$.

▶ **Theorem 11.** *Let V be a scanning function in* PSPACE *that is g-filling for a polynomial bound g. If Z is polynomial space random, then so is $Z \circ V$.*

Proof. Suppose $Z \circ V$ is not polynomial space random. Let $G = (V, B)$ be a betting strategy in PSPACE that succeeds on Z; thus, B succeeds on $Z \circ V$.

We define a martingale D in PSPACE that succeeds on Z. We may assume that $g(n) \geq n$. For $t \geq g(|w|)$ let

$$D(w) = 2^{|w|-t} \sum_{|\alpha|=t \ \wedge \ \alpha \sim_V w} B(\alpha).$$

By the claim above and since B is a martingale, this definition is independent of t. Note that among the runs α of length t, a fraction of $2^{-|w|}$ satisfy that $\alpha \sim_V w$; so $D(w)$ is simply the average value of $B(\alpha)$ over all such α.

If we let $t = g(|w|)$, by the hypotheses that G is in PSPACE and that g is a polynomial, D is in PSPACE.

The rest of the argument somewhat simplifies the one of [3] in the present context.

▶ **Lemma 12.** *D is a martingale.*

Let w be a string of length n. If $|\alpha| = g(n+1)$ and $\alpha \sim_V w$, then either $\alpha \sim_V w0$ or $\alpha \sim_V w1$. Letting $u = g(n+1)$, for each $r = 0, 1$ we have

$$D(wr) = 2^{|w|+1-u} \textstyle\sum_{|\alpha|=u \ \wedge \ \alpha \sim_V wr} B(\alpha).$$

Hence, since the definition of $D(w)$ does not depend on the choice of $t \geq g(|w|)$,

$$D(w0) + D(w1) = 2^{|w|+1-u} \textstyle\sum_{|\alpha|=u \ \wedge \ \alpha \sim_V w} B(\alpha) = 2D(w).$$

▶ **Lemma 13.** *D succeeds on Z.*

We may assume that $B(x) > 0$ for each x. The Savings Lemma (see e.g. Nies [11, 7.1.14]) states that each computable martingale M can be turned into a computable martingale $\widehat{M}$ that succeeds on the same sets, and has the extra property that $\widehat{M}(\beta) \geq \widehat{M}(\alpha) - 2$ for each strings $\beta \supseteq \alpha$ (namely, $\widehat{M}$ never loses more than 2). It is easy to see from the proof that if M is computable in polynomial space, then so is $\widehat{M}$. So we may assume that B has this property.

This implies that for each $c \in \mathbb{N}$ there is a prefix α of $Z \circ V$ such that

$$B(\beta) \geq c \text{ for each string } \beta \succeq \alpha.$$

By definition of $Z \circ V$ we have $\alpha(i) = Z(V(\alpha \upharpoonright i))$ for each $i < |\alpha|$. Let $r = 1 + \max_{i<|\alpha|} V(\alpha \upharpoonright i)$ be 1+ the maximum query asked in the run of V on α, and let $w = Z \upharpoonright r$. So $g(r) \geq |\alpha|$.

If $\beta \sim_V w$ is a string such that $|\beta| = g(r)$, then $\beta \succeq \alpha$, for $\alpha(r) \neq \beta(r)$ for some $r < |\alpha|$ would imply that $\beta \not\sim_V w$ as w answers all such queries correctly. So $B(\beta) \geq c$. Hence $D(w) \geq c$ because $D(w)$ is the average over values $B(\beta)$ for all such β. ◀

▶ **Corollary 14.** *Let S be a polynomial time computable permutation of $\{0\}^*$ such that S^{-1} is polynomially bounded. If Z is polynomial space random, then so is $Z \circ S$.*

Proof. The permutation S can be viewed as a scanning function V_S that only looks at the length of the input: $V_S(\alpha) = S(|\alpha|)$. By hypothesis on S, the scanning function V_S is polynomially filling. So $Z \circ S = Z \circ V_S$ is polynomial space random by the theorem. ◀

The foregoing corollary can be restated in terms of randomness on languages in the sense of [1]: Let S be a exponential time computable permutation of $\{0,1\}^*$ such that $|S^{-1}(x)| = O(|x|)$ for each string x. If a language Z is exponential space random, then so is $Z \circ S$.

We end with a question. Recall that PP denotes probabilistic polynomial time, a subclass of PSPACE. If P = PP, is polynomial time randomness closed under permutations S of $\{0\}^*$ such that S, S^{-1} are polynomial time computable?

References

1 K. Ambos-Spies and E. Mayordomo. Resource-bounded measure and randomness. *Lecture Notes in Pure and Applied Mathematics*, pages 1–48, 1997.

2 V. Brattka, J. Miller, and A. Nies. Randomness and differentiability. *Transactions of the AMS*, 368:581–605, 2016. arXiv version at arxiv.org/abs/1104.4465.

3 H. Buhrman, D. Van Melkebeek, K. Regan, D. Sivakumar, and M. Strauss. A generalization of resource-bounded measure, with application to the BPP vs. EXP problem. *SIAM Journal on Computing*, 30(2):576–601, 2000.

4 R. Downey and D. Hirschfeldt. *Algorithmic randomness and complexity*. Springer-Verlag, Berlin, 2010. 855 pages.

5 S. Figueira and A. Nies. Feasible analysis, randomness, and base invariance. *Theory of Computing Systems*, 56(3):439–464, 2015.

6 D. Hirschfeldt, A. Nies, and F. Stephan. Using random sets as oracles. *J. Lond. Math. Soc. (2)*, 75(3):610–622, 2007.

7 B. Kjos-Hanssen, P. Nguyen, and J. Rute. Algorithmic randomness for doob's martingale convergence theorem in continuous time. *arXiv preprint arXiv:1411.0186*, 2014.

8 M. Li and P. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Graduate Texts in Computer Science. Springer-Verlag, New York, second edition, 1997.

9 P. Martin-Löf. The definition of random sequences. *Inform. and Control*, 9:602–619, 1966.

10 A. Nies. Lowness properties and randomness. *Advances in Mathematics*, 197:274–305, 2005.

11 A. Nies. *Computability and Randomness*, volume 51 of *Oxford Logic Guides*. Oxford University Press, Oxford, 2009. 444 pages. Paperback version 2011. doi:10.1093/acprof:oso/9780199230761.001.0001.

12 André Nies. Differentiability of polynomial time computable functions. In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, volume 25 of *LIPIcs*, pages 602–613. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.602.

13 C.P. Schnorr. *Zufälligkeit und Wahrscheinlichkeit. Eine algorithmische Begründung der Wahrscheinlichkeitstheorie*. Springer-Verlag, Berlin, 1971. Lecture Notes in Mathematics, Vol. 218.

14 Michiel van Lambalgen. The axiomatization of randomness. *J. Symbolic Logic*, 55(3):1143–1167, 1990.

15 Y. Wang. *Randomness and Complexity*. PhD dissertation, University of Heidelberg, 1996.

Succinct Oblivious RAM

Taku Onodera
Human Genome Center, Institute of Medical Science, The University of Tokyo, Tokyo, Japan

Tetsuo Shibuya
Human Genome Center, Institute of Medical Science, The University of Tokyo, Tokyo, Japan

Abstract

As online storage services become increasingly common, it is important that users' private information is protected from database access pattern analyses. Oblivious RAM (ORAM) is a cryptographic primitive that enables users to perform arbitrary database accesses without revealing any information about the access pattern to the server. Previous ORAM studies focused mostly on reducing the access overhead. Consequently, the access overhead of the state-of-the-art ORAM constructions are almost at practical levels in certain application scenarios such as secure processors. However, we assume that the server space usage could become a new important issue in the coming big-data era. To enable large-scale computation in security-aware settings, it is necessary to rethink the ORAM server space cost using big-data standards.

In this paper, we introduce "succinctness" as a theoretically tractable and practically relevant criterion of the ORAM server space efficiency in the big-data era. We, then, propose two succinct ORAM constructions that also exhibit state-of-the-art performance in terms of the bandwidth blowup and the user space. We also give non-asymptotic analyses and simulation results which indicate that the proposed ORAM constructions are practically effective.

2012 ACM Subject Classification Theory of computation → Cryptographic protocols, Security and privacy → Management and querying of encrypted data

Keywords and phrases Oblivious RAM, Succinct Data Structure, Balls-into-bins

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.52

Funding This work was supported by JSPS KAKENHI Grant Number 17H01693, 17K20023JST and CREST Grant Number JPMJCR1402.

Acknowledgements We thank Paul Sheridan for helpful discussion.

1 Introduction

Oblivious RAM (ORAM) is a cryptographic primitive that enables users to access a database on a server without revealing the access pattern to the server. Though originally introduced for software protection [14], ORAM is directly relevant to the present cloud computing.

In the previous studies on ORAM, researchers focused mainly on reducing the access bandwidth cost, a performance measure used as a proxy of the access time. This is because even the current most state-of-the-art ORAM constructions have two or three orders of magnitude larger bandwidth cost than the ordinary (non-secure) accesses. However, in certain settings, the ORAM access is already rather efficient. For example, Maas et al. proposed PHANTOM [23], an ORAM-based secure processor, and reported that if PHANTOM is deployed on the server, SQLite queries can be performed without revealing the access pattern at the cost of 1.2–6× slowdown compared to non-secure SQLite queries. In such cases, it is reasonable to pay more attention to performance measures other than the access speed.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 52; pp. 52:1–52:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


In particular, the *server space usage* is a very important performance measure for big-data applications. First, there are applications where the amount of data is virtually unbounded, and thus the limit of the available space defines the limit of the analyses. Second, due to the cache effect, small memory usage often leads to faster computation. Third, space costs money, especially in a cloud computing server. The second and the third points are especially relevant if the data is meant to be stored in the main memory (by default), which is exactly the case in ORAM application scenarios such as PHANTOM.

In most modern ORAM constructions, if the size of the original database is n bits, the amount of the space required by the server is $n + \Theta(n)$ bits. In this paper, we investigate the possibility of ORAM constructions that need only $n + o(n)$ bits of server space. We call such ORAM constructions *succinct*. This space efficiency formalization is widely used in the field of *succinct data structures* and has proved to be useful to design practically relevant space-efficient data structures in theoretically clean ways.

The main difficulty to achieve succinctness is that most existing ORAM construction approaches rely on the use of linear amount of "dummy" data. The situation is similar to conventional hash tables, which need extra space linear to the stored keys size. Although it seems possible to reduce the constant factor of the extra space to some extent, it is not at all trivial if one can achieve sublinear extra space maintaining the state-of-the-art performance in other aspects such as access bandwidth and user space usage.

Results. Table 1 shows the performance of the proposed methods and the existing methods. Our first construction takes $n(1 + \Theta(\frac{\log n}{B} + \frac{g(n)}{f_1(n)/\log n}))$-bit server space where n is the database size, $f_1(\cdot)$ is a function such that $f_1(n) = \omega(\log n)$ and $O(\log^2 n)$, $g(\cdot)$ is a function such that $g(n) = \omega(1)$ and $o(\sqrt{f_1(n)/\log n})$, and B is the size of a *block*, the unit of communication between the user and the server. The bandwidth blowup is $O(\log^2 n)$ and the user space is $O(f_1(n))$ blocks. Our second construction has $n(1+\Theta(\frac{\log n}{B} + \frac{\log\log n}{f_2(n)}))$-bit server space, $O(\log^2 n)$-bandwidth blowup and $O(f_2(n) + R(n))$-user space where $f_2(\cdot)$ is a function such that $f_2(n) = \omega(\log\log n)$ and $O(\log^2 n)$, $R(\cdot)$ is a function with $R(n) = \omega(\log n)$.

For example, if $B = \log^2 n$, $R = \log n \log\log n$, $f_1(n) = f_2(n) = \log n \log\log n$ and $g(n) = \log\log\log n$, the user space is $O(\log n \log\log n)$ in both constructions and the server space is $n(1+\Theta(\frac{\log\log\log n}{\log\log n}))$ (resp. $n(1+\Theta(\frac{1}{\log n}))$) bits in the first (resp. second) construction.

The second construction has better theoretical performance than the first one. However, in practice, with some parameter settings, the first construction also works comparably well as the second construction depending on which performance measure one cares (See Section 5). The first construction is also the basis of the second construction.

If $B = \omega(\log n)$, Goldreich's construction [14] and our constructions are succinct. (Each of these methods works as long as $B \geq c \lg n$ for c around 3.) The assumption $B = \omega(\log n)$ is justified as follows. Stefanov et al. [38] mentioned that the typical block size is 64–256 KB (resp. from 128B to 4KB) in cloud computing scenario (resp. software protection scenario). Even $B \geq \lg^{1.5} n$ holds if $n \leq 2^{6501}$ (resp. $n \leq 2^{97}$) in cloud computing (resp. software protection) scenario with moderate block size of 64KB (resp. 128B).

We achieved exponentially smaller bandwidth blowup compared to Goldreich's construction [14], which is the only preceding non-trivial succinct ORAM construction.

The bandwidth blowup of our constructions are smaller or equal to other non-succinct constructions except [22], [7] and [37]. [22] is based on a very expensive procedure called oblivious sorting and the constant factor of the bandwidth blowup is prohibitively large. [7] has $O(1)$-bandwidth blowup but it requires several assumptions. First, the server needs to perform some computation, e.g., homomorphic encryption evaluation. (In every other

Table 1 Comparison of theoretical performance. Bandwidth blowup is the number of blocks required to be communicated for accessing one block of data. User space includes the temporary space needed during access procedures. n is the database size in bits and B is the block size in bits. B must satisfy $B \geq c_1 \lg n$ and $B = O(n^{c_2})$ for constants $c_1 > 1$, $0 < c_2 < 1$. Typically, c_1 is around 3. $f_1(\cdot)$ is an arbitrary function such that $f_1(n) = \omega(\log n)$ and $O(\log^2 n)$. $f_2(\cdot)$ is an arbitrary function such that $f_2(n) = \omega(\log \log n)$ and $O(\log^2 n)$. $R(\cdot)$ is an arbitrary function such that $R(n) = \omega(\log n)$. $g(\cdot)$ is an arbitrary function such that $g(n) = \omega(1)$ and $o(\sqrt{f_1(n)/\log n})$. Bounds with † are amortized. The method in [7] requires additional assumptions.

	Server space (#bits)	Bandwidth blowup	User space (#block)
Goldreich [14]	$n(1+\Theta(\frac{\log n}{B}+\frac{1}{\sqrt{n}}))$	$O(\sqrt{n}\log n)^{\dagger}$	$O(1)$
Kushilevitz, et al. [22]	$n(1+\Theta(1))$	$O(\frac{\log^2 n}{\log\log n})$	$O(1)$
Stefanov, Shi, Song [37]	$n(1+\Theta(1))$	$O(\log n)$	$O(n)$
Stefanov et al. [38]	$n(1+\Theta(1))$	$O(\log^2 n)$	$O(R(n))$
Devadas et al. [7]	$n(1+\Theta(1))$	$O(1)$	$O(1)$
Our result (Theorem 3)	$n(1+\Theta(\frac{\log n}{B}+\frac{g(n)}{f_1(n)/\log n}))$	$O(\log^2 n)$	$O(f_1(n))$
Our result (Theorem 5)	$n(1+\Theta(\frac{\log n}{B}+\frac{\log\log n}{f_2(n)}))$	$O(\log^2 n)$	$O(f_2(n)+R(n))$

construction in Table 1, the server suffices to respond to read/write requests.) [7] also requires a computational assumption (decisional composite residuosity or learning with errors assumption), and larger block size ($B = \widetilde{\omega}(\log^2 n)$ to $\widetilde{\omega}(\log^6 n)$ depending on the case, where $\widetilde{\omega}(\cdot)$ hides a polyloglog factor). [37] takes cn-bit user space where $c \ll 1$. This method is effective for ordinary cloud computing setting but the user space is too large for secure processor setting — the PHANTOM-like applications where server space efficiency is more important.

Possible applications. There are several ORAM application scenarios with different requirements. Our methods are particularly relevant to *secure processor* scenario. In this scenario, it is assumed that a special processor under the control of the user is available in a remote server and the adversary cannot observe the activities inside the processor. The cloud service user sends a piece of code to the trusted processor, which, in turn, executes the code on the server. The communication between the cloud service user and the secure processor is protected by private key encryption. ORAM is implemented inside of the trusted processor using FPGA and it hides the processor's access pattern to the main memory on the server. After executing the code, the secure processor may return the (encrypted) output to the cloud service user. One of the main advantages of this approach over the conventional ORAM application, in which the cloud service user locally executes ORAM, is that ORAM bandwidth blowup applies to the relatively cheap processor–memory communication rather than the costly over-network communication. Note that, with the ORAM user-server terminology, the secure processor (resp. the main memory) is the user (resp. the server).

In secure processor scenario,

- the user space is very limited, e.g., 6MB;
- The server usually does not perform complex computation;
- Simple ORAM algorithms are desirable for hardware implementation;
- The server space is much larger than the user space but there is some noticeable limit. The server can use disks if needed but it greatly slows down accesses.

In most existing secure processor systems, the Path ORAM [38] or its close variants are used [11, 23, 33, 12]. Indeed, the Path ORAM satisfies the first three requirements above.

However, it does not capture the last one. For example, suppose 128GB database is stored in the Path ORAM. If the block size is 128B, it takes about 10G blocks, i.e., 1.28TB (to ensure rigorous security). Then, each ORAM access procedure takes about 31μs assuming each memory access takes 100ns. If half of the 10G blocks are stored in the main memory and the other half is stored in the disk, due to the randomized access pattern of the Path ORAM, almost every ORAM access procedure ends up a disk seek, which takes milliseconds order time. In such cases, it is reasonable to use another ORAM construction that takes, say, half the space of the Path ORAM even though it requires twice as many memory accesses.

Tree-based ORAM. Our ORAM constructions are tree-based. In a typical tree-based ORAM construction, N blocks are stored in a complete binary tree with N leaves on the server. Each node of the tree can store up to Z blocks where Z is a constant. Each block is assigned a position label, a uniformly random integer in $[N]$. A block with position label i must be stored at some node on the path from the root to the i-th leaf. This framework was introduced by Shi et al. [36] and used in many subsequent studies [38, 13, 33, 5, 7].

Consider a particular block b. As the user continuously issues access requests, b moves around the tree in roughly the following manner. First, when the user issues an access request to b, b is picked out of the tree and given a new uniformly random position label. Then, b is inserted into the tree from the root. If the user issues an access request to another block, then, with some probability, b will move down the path to the leaf indicated by its position label. If the next node on the path is full, b must wait for the blocks "ahead" to move down. If the pace at which the blocks move down the tree cannot keep up with the pace at which blocks are picked out and reinserted from the root, then, some blocks will not be able to reenter the tree. If such "congestion" occurs, the user must maintain the overflown blocks locally.

Note that most space in the tree is wasted: there are $2N - 1$ nodes in the tree, each with capacity Z, whereas there are only N blocks. Thus, to save server space, it is desirable to make the tree more compact, for example, by reducing Z. However, to maintain a low probability of "congestion", it is desirable to make the tree larger, for example, by increasing Z. To construct a succinct tree-based ORAM, we need to satisfy these conflicting demands.

Our ideas. One of our key ideas is the following two-stage tree layout. We first change the tree to a complete binary tree with $N/\lg^{1.4} N$ leaves (assume this is a power of 2). In addition, we set the capacity of each leaf node to $\lg^{1.4} N + \lg^{1.3} N$ while keeping the capacity of each internal node at Z. The total size of the leaf nodes is then $N + N/\lg^{0.1} N$, and the total size of all tree nodes except the leaves is $\Theta(N/\lg^{1.4} N)$. Thus, the total size of the entire tree is $N + o(N)$. We choose each position label from $[N/\lg^{1.4} N]$.

To see why blocks can flow around in this tree without much congestion, suppose that the user inserts each block directly into the leaf node pointed to by the block's position label. Clearly, the loads of leaves in this hypothetical setting dominates the loads of leaves in the real setting. Then, the situation would exactly be the same as the "balls-into-bins" game [24] with N balls and $N/\lg^{1.4} N$ bins. In particular, the number of blocks stored in each leaf node is $\log^{1.4} N + \Theta(\log^{1.2} N)$ with high probability. Thus, every leaf node has sufficient capacity to store all of its assigned blocks. Furthermore, the blocks in the internal nodes flow as smoothly as in the original non-succinct ORAM construction since we did not modify that part. Thus, the blocks flow without much congestion in the tree.

Another key idea follows naturally from the above argument, specifically from the connection to the balls-into-bins game. A remarkable phenomenon known as "the power

of two choices" states that, in the balls-into-bins game, if one chooses two bins uniformly and independently for each ball, and throws the ball into the least loaded bin, the bin loads will be distributed much more tightly around the mean than they are in the one-choice game [1, 3, 24]. The maximum bin load corresponds to the leaf node size in tree-based ORAM constructions. Thus, the size of the tree can be further decreased by using the two-choice strategy to assign the position labels. This is the idea behind the construction in Section 4.

We note that the current paper is the first to apply the power of two choices to tree-based ORAM. (Some non-tree-based constructions [30, 16, 22] use the two choices idea in the form of cuckoo hashing [29].) Moreover, the resulting algorithms keep the simplicity of the Path ORAM [38], which is a highly valuable asset in the relevant application scenario as mentioned above. As for the analysis, the existing stash size analyses [38, 33] do not seem to work with parameter regimes required for succinctness. We will give a different proof route (though it still heavily borrows from [38, 33]) in the full version of the paper.

Our contributions. Our contributions in the current paper are as follows:

- We introduce the notion of succinct oblivious RAM. This is a promising first step to systematically design ORAM constructions with small server space usage;
- We propose two succinct ORAM constructions. Not only being succinct, these constructions exhibit state-of-the-art performance in terms of the bandwidth blowup. The methods are simple and easy to implement.
- We also give non-asymptotic bounds and simulation results which indicate that the proposed methods are practically effective.

Related work. In the field of succinct data structures [20, 19], the goal is to represent an object such as a string [26, 34, 17, 9, 15, 35, 21, 10, 18, 27] or a tree [6, 25, 31, 2, 8, 28] in such a way that a) only $OPT + o(OPT)$ bits are required, and b) relevant queries such as random access or substring search are efficiently supported. Here, OPT is the information theoretic optimum, i.e., the minimum number of bits needed to represent the object.

The current study is related to succinct data structures in the following way. Suppose a remote server hosts a database that is implemented by a succinct data structure, and a user wishes to access the database without revealing the access pattern to the server. The user, of course, can apply any existing ORAM constructions. However, if ORAM increases the database size by some constant factor, it destroys the $OPT + o(OPT)$ bound guaranteed by the succinct data structure. One can apply the succinct ORAM constructions proposed in this paper to hide succinct data structure access pattern on a remote storage device without harming the theoretical guarantee on the data structure size.

Notations. We denote the set $\{0, 1, \ldots n-1\}$ as $[n]$ for a non-negative integer n. We write $\lg x$ to denote the base-2 logarithm of x and $\ln x$ to denote the natural logarithm of x. We write $\log x$ to denote the logarithm of x in the context where the base can be any positive constant. We write $\mathrm{poly}(n)$ to denote n^c for some constant $c > 0$. A negligible function of n is defined to be a function that is asymptotically smaller than $1/n^c$ for any constant $c > 0$.

2 ORAM: Preliminaries

Definition. Suppose there are three parties the *user*, the *server* and the *oblivious RAM (ORAM) simulator*. Let each of B and n be a positive integer and $N := n/B$. (We assume n is a multiple of B for brevity.) The value B models the unit of communication and n

models the database size. We call a chunk of B bits a *block*. A *logical (resp. physical) access request* is a triplet $(\mathrm{op}, \mathrm{addr}, \mathrm{val})$, where $\mathrm{op} \in \{\mathrm{read}, \mathrm{write}\}$, $\mathrm{addr} \in [N]$ (resp. $\mathrm{addr} \in \mathbb{N}$), $\mathrm{val} \in \{0,1\}^B$. The user sends logical access requests to the ORAM simulator and receives a block for each request. The server receives physical access requests from the ORAM simulator and returns a block for each request in the following way: for (read, i, v), the server returns v of the most recent request (write, i, v). The ORAM simulator takes a sequence of logical access requests from the user and for each logical access request, it makes a sequence of physical access requests to the server receiving a returned block for each of them, and returns a block to the user. The ORAM simulator is possibly stateful and probabilistic. It must respond to logical access requests online and must satisfy the following conditions:

Correctness The ORAM simulator is correct iff, for a logical access request with $\mathrm{addr} = i$, it returns v of the previous and most recent logical access request (write, i, v);[1]

Security The ORAM simulator is computationally (resp. information theoretically) secure iff, for any logical access request sequences of the same length, the distributions of the addr values of the resulting physical access requests are computationally (resp. information theoretically) indistinguishable.

An ORAM construction is an ORAM simulator implementation. We have distinguished the user from the ORAM simulator for exposition but in practice, an ORAM simulator is a program run by the user. Thus, we do not distinguish them in the rest of the paper.

Encryption. In the ORAM constructions considered in this paper, the user holds a symmetric cipher key and every block is encrypted when it is stored on the server. Though encryption increases the database size, the increase is minor[2]and we ignore the space blowup due to encryption in the rest of the paper.

Performance measures. The most popular ORAM performance measures include the space required by the user/server and time required for each logical access.

In most ORAM constructions, the user needs to maintain a small amount of information locally. In addition to this, in some constructions, the user temporarily need to store more information during the access procedure. We refer the amount of the space the user temporarily needs during access procedure as *temporary space usage* and the amount of the space the user needs even if no access is made as *permanent space usage*.

In this paper, we pay special attention to the server space usage. In particular, we use the following notion of *succinctness* as a criterion for ORAM server-space efficiency:

▶ **Definition 1.** If the server space usage of an ORAM construction representing an n-bit database is $n + o(n)$ bits, the ORAM construction is said to be succinct.

As for the access efficiency, following the previous studies, we use the amount of communication between the user and the server as a proxy for the access time. We define the *bandwidth blowup* of an ORAM construction to be the number of blocks that needs to be communicated between the user and the server per logical access. In other words, the bandwidth blowup is the ratio of communication amount needed for secure access to communication amount needed for ordinary (insecure) access.

---

[1] We use the convention that not only read but also write requests have return values.

[2] In theory, we can guarantee the semantic security and succinctness at the same time with extra bits of amount $\omega(\log n)$ and $o(B)$ per each block. In practice, assuming that we use "counter mode" block cipher with 128 bits counters and the typical block sizes mentioned in Section 1, the space blowup is 1/4096–1/16384 (resp. 1/8–1/256) factor in cloud computing (resp. software protection) scenario.

Asymptotic behavior of parameters. Among the ORAM-related parameters, the original database size n and block size B are outside of the user's control. Other parameters, e.g., the metadata size, can be chosen by the user. We assume that B is a function of n satisfying $B = \omega(\log n)$. (See Section 1 for the justification.) Thus, after all, n is the only free parameter on which the other parameters depend. In all asymptotic statements in this paper, the limit is taken as $n \to \infty$.

Sub-ORAM. We use an ORAM construction encapsulated into the following proposition as a blackbox. Concretely, the Path ORAM [38] suffices.

▶ **Proposition 2.** *Let n be the database size and B be the block size, in bits. If $B \geq 3 \lg n$ and $B = O(n^c)$ for some $0 < c < 1$, there exists an information theoretically secure ORAM construction such that i) the server's space usage is $n(10 + \Theta(\frac{\log n}{B}))$ bits; ii) the worst-case bandwidth blowup is $O(\log^2 n)$; iii) the user's temporary space usage is $O(\log n)$ blocks; and iv) for any $R = \omega(\log n)$, the probability that the user's permanent space usage becomes larger than R blocks during $\mathrm{poly}(n)$ logical accesses is negligible.*

3 Succinct ORAM Construction

In this section, we prove the following theorem.

▶ **Theorem 3.** *Let n be the database size and B be the block size, both in bits. If $B \geq 3 \lg n$ and $B = O(n^c)$ for some constant $0 < c < 1$, then for any $f : \mathbb{N} \to \mathbb{R}$ such that $f(n) = \omega(\log n)$ and $f(n) = O(\log^2 n)$ and any $g : \mathbb{N} \to \mathbb{R}$ such that $g(n) = \omega(1)$ and $g(n) = o(\sqrt{f(n)/\log n})$, there exists an information theoretically secure ORAM construction such that i) the server's space usage is bounded by $n(1 + \Theta(\frac{\log n}{B} + \frac{g(n)}{\sqrt{f(n)/\log n}}))$ bits; ii) the worst case bandwidth blowup is $O(\log^2 n)$; iii) the user's temporary space usage is $O(f(n))$ blocks; and iv) for any $R = \omega(\log n)$, the probability that the user's permanent space usage becomes larger than R blocks during $\mathrm{poly}(n)$ logical accesses is negligible.*

▶ **Corollary 4.** *If $B = \omega(\log n)$, then, the ORAM construction of Theorem 3 is succinct.*

3.1 Description

For the clarity of explanation, we first describe a simplified ORAM construction where the user needs to maintain a large amount of information locally. Then, we obtain an ORAM construction with the claimed bounds by slightly modifying the simplified construction.

As we mentioned in Section 1, in a tree-based ORAM construction, blocks on the server are stored in the nodes of a complete binary tree. The key point of the method in this section is the choice of the tree height L and the leaf node capacity M. Specifically, in the rest of this section, let $L := \lceil \lg \frac{N}{f(n)} \rceil$ and $M := \lceil \frac{N}{2^L} + g(n)\sqrt{\frac{NL}{2^L}} \rceil$ where $N := n/B$. We assume, for brevity, that each of $\lg \frac{N}{f(n)}$ and $\frac{N}{2^L} + g(n)\sqrt{\frac{NL}{2^L}}$ is an integer.

Block usage. The ORAM is supposed to provide the user with an interface to access the database as if it is stored in array A of B-bit blocks (Section 2). We use blocks as follows :

- Each block is either a *data block* or a *metadata block*;
- Each data block is either a *real block* or a *dummy block*. A real block contains an entry of A. A dummy block does not contain any information on the database contents and is used only to hide the access pattern;

- Each real block is given a *position label*, a value in $[2^L]$;
- A metadata block contains the metadata of several data blocks. For each data block, its metadata consists of
  - type: A flag indicating whether the block is real or dummy;
  - addr: If the block is real and represents $A[i]$, the value of addr is i. If the block is a dummy, the value is arbitrary;
  - pos: If the block is real with position label i, the value of pos is i. If the block is a dummy, the value is arbitrary.

Data layout. The server maintains a tree containing data blocks, which we call *data tree*, and another tree containing metadata blocks, which we call *metadata tree*. The data tree is used in such a way that at each point of time, it contains most real blocks with high probability. The user maintains *stash*, which contains the real blocks that are not in the data tree, and *position table*, which contains the position labels of all real blocks.

The data tree is a complete binary tree with 2^L leaves. Each node of the tree is a *bucket*, which is a container that can accommodate a certain number of blocks. We call the buckets corresponding to the internal nodes as *internal buckets* and the buckets corresponding to the leaf nodes as *leaf buckets*. The size of each internal bucket is Z (blocks) while the size of each leaf bucket is M (blocks). We will determine Z to be 3 in the full version of the paper but for now, we consider it as an arbitrary constant. The data tree is represented as the bitstring derived by concatenating all buckets in breadth first order. As is well-known, with this representation, given an index of a node, the index of the parent or left/right child can be derived by simple arithmetic. The total space usage of the data tree is equal to the sum of the bucket sizes.

The metadata tree is also a complete binary tree with 2^L leaves. Each node of the tree is the metadata of the data blocks in the corresponding bucket of the data tree. The metadata tree is represented similarly to the data tree but there is a subtlety. If the metadata of the blocks in a bucket has a size smaller than B, it is wasteful to allocate one full block for them. To avoid this waste, we represent metadata tree as the bitstring derived by concatenating the metadata of all data blocks in the data tree in breadth first order. The space usage of the metadata tree is equal to the sum of all metadata of all data blocks.

Each real block in the stash is maintained with its addr and pos. The stash can be any linear-space data structure that efficiently supports insertion, deletion and range query by pos, e.g., a self balancing binary search tree.

The position table stores the position label of real block storing $A[i]$ in the i-th entry.

Access procedure. Access requests are processed in such a way that the following invariant conditions are always satisfied:

- Each real block is stored either in the data tree or in the stash;
- If a real block with position label ℓ is stored in the data tree, it is in the bucket on the path from the root to the ℓ-th leaf.

Below, we give a high-level description of the main routine and we will provide the pseudocode in the full version of the paper. To read the explanation here, it should suffice to know that $P(\ell)$ means the path from the root to the ℓ-th leaf of the data tree.

Let b_a be the accessed block. We first read the position label ℓ of b_a from the position table and update the position table entry to a number chosen uniformly at random from $[L]$, which will become the new position label of b_a after the access operation is finished. By the invariant conditions above, b_a is either in the stash or $P(\ell)$. We scan $P(\ell)$ and retrieve b_a

if it is in $P(\ell)$. If b_a was not in $P(\ell)$, we retrieve it from the stash. If the current request is a write request, we update the block contents to the new value. Then, we insert b_a with the updated position label and the possibly updated value into the stash. After that, we perform EvictPath operation. The purpose of this operation is a) to move back the blocks in the stash into the tree and b) to move the real blocks in the tree downwards (far from the root). To do this, EvictPath retrieves all real blocks in the path $P(\text{BitReversal}(\text{G}))$ (to be explained shortly) into the stash and then, going up $P(\text{BitReversal}(\text{G}))$ from leaf to the root, tries to move as many blocks in the stash into the buckets on the path. If some blocks are left in the stash after EvictPath, the user keeps them charging the permanent space usage. Lastly, the value stored at b_a is returned.

The function BitReversal$(\cdot)$ takes an L-bit integer x and returns the bit reversed version of x while G is the number of Access operations called so far (modulo 2^L). This BitReversal-based scheduling of EvictPath was first proposed by Gentry et al. [13] and is advantageous to keep the stash size small. It also enables to simplify stash size analysis, which we will provide in the full version. Here, it suffices to note that G (and BitReversal(G)) is independent of the accessed database locations.

Outsourcing position table. In the construction described so far, the user space usage is much larger than the bound claimed in Theorem 3 since the user needs to maintain the position table locally. To obtain Theorem 3, we modify the construction so that the position table is stored on the server using the Path ORAM [38]. Accesses to position tables are replaced by a Path ORAM write.

3.2 Analysis

Security. Fix $t > 0$. Let $\mathbf{a}$ be a length $t > 0$ sequence of logical addresses to be accessed and $\mathbf{a}'$ be the corresponding sequence of physical addresses (indices of the server memory) to be accessed. The sequence $\mathbf{a}'$ is determined by $\mathbf{a}$ and the randomness used by the ORAM simulator. To prove the information theoretic security, it suffices to show that $\mathbf{a}'$ really does not depend on $\mathbf{a}$. The sequence $\mathbf{a}'$ consists of $\mathbf{a}'_1$, the physical addresses accessed in the recursive access call to the Path ORAM and $\mathbf{a}'_2$, those accessed in the rest parts. The addresses $\mathbf{a}'_1$ is determined by the Path ORAM access procedure and is independent of $\mathbf{a}$ due to the information theoretic security of the Path ORAM. The addresses $\mathbf{a}'_2$ consists of addresses accessed by ReadPath(ℓ, a) and EvictPath$()$. ReadPath(ℓ, a) accesses the path $P(\ell)$, which is determined by ℓ, the position label of the accessed block. Since the position labels are chosen independently and uniformly at random, the ReadPath accesses are independent of $\mathbf{a}$. EvictPath accesses $P(\text{BitReversal}(G))$, which is determined by G, the number of times Access was called (modulo 2^L). Thus, the accesses of EvictPath is also independent of $\mathbf{a}$. Therefore, $\mathbf{a}'$ is independent of $\mathbf{a}$.

Server space. First, it is helpful to observe that $\log N = \Theta(\log n)$, $L = \Theta(\log n)$ and $M = \Theta(f(n))$. Remember that the server holds the data tree, the metadata tree and the position table. The total size of the internal (resp. leaf) buckets is $Z(2^L - 1)$ (resp. $M2^L$) blocks. Since $Z(2^L - 1) < Z2^L = ZN/f(n)$ and $M2^L = N + g(n)\sqrt{NL2^L} = N(1 + \Theta(\frac{g(n)}{\sqrt{f(n)/\log n}})) = N(1 + \Theta(h(n)))$ where $h(n) := \frac{g(n)}{\sqrt{f(n)/\log n}}$, the number of the blocks in the data tree is bounded by $ZN/f(n) + N(1 + \Theta(h(n))) = N(1 + \Theta(\frac{1}{f(n)} + h(n)))$.

The metadata for each data block takes 1 bit for type, $\lceil \lg N \rceil$ bits for addr and L bits for pos. The total is $\Theta(\log n)$ bits $= \Theta(\frac{\log n}{B})$ blocks. Thus, the number of bits in the data tree and the

metadata tree combined is $BN(1+\Theta(\frac{1}{f(n)}+h(n)))(1+\Theta(\frac{\log n}{B})) = n(1+\Theta(\frac{\log n}{B}+h(n)))$. The position labels take $NL = n\frac{L}{B} \leq n\frac{\lg n}{B}$ bits. By Proposition 2, the Path ORAM containing the position table takes $\Theta(n\frac{\log n}{B})$ bits. Thus, the server space is $n(1+\Theta(\frac{\log n}{B}+h(n)))$ bits.

Bandwidth blowup. The bandwidth cost of each of ReadPath and EvictPath is proportional to the sum of the numbers of the blocks in a root–leaf path in the data tree and the metadata tree. The number for the data tree is $ZL + M = O(\log n) + O(f(n)) = O(f(n))$. The number for the metadata tree is around $\frac{2\lg N+1}{B} = o(1)$ factor of that for the data tree. The bandwidth cost for accessing the position table is $O(\log^2 n)$ by Proposition 2. Therefore, the bandwidth blowup of Access is $O(\log^2 n)$.

User space. The temporary user space usage is proportional to the sum of the numbers of the blocks in a root–leaf path in the data tree and the metadata tree. As is shown in the bandwidth analysis, the latter is bounded by $O(f(n))$. We prove the bound on the permanent user space usage, i.e., the stash size in the full version of the paper.

4 Succincter ORAM Construction

In this section, we prove the following theorem.

▶ **Theorem 5.** *Let n be the database size and B be the block size, both in bits. If $B \geq 3\lg n$ and $B = O(n^c)$ for some $0 < c < 1$, then for any $f : \mathbb{N} \to \mathbb{R}$ such that $f(n) = \omega(\log\log n)$ and $f(n) = O(\log^2 n)$, there exists an information theoretically secure ORAM construction for which i) the server's space usage is bounded by $n(1+\Theta(\frac{\log n}{B}+\frac{\log\log n}{f(n)}))$ bits; ii) the worst case bandwidth blowup is $O(\log^2 n)$; iii) the user's temporary space usage is $O(\log n + f(n))$ blocks; and iv) for any $R = \omega(\log n)$, the probability that the user's permanent space usage becomes larger than R blocks during $\mathrm{poly}(n)$ logical accesses is $n^{-\omega(1)}$.*

▶ **Corollary 6.** *If $B = \omega(\log n)$, then, the ORAM construction of Theorem 5 is succinct.*

4.1 Description

As in Section 3, we first explain a simplified version with a large user space usage, and construct the full version that achieves the claimed bounds from the simplified version.

Let $L := \lceil \lg(N/f(n)) \rceil$ and $M := \lceil N/2^L + (1+\varepsilon)\lg L \rceil$ where $N := n/B$ and $\varepsilon > 0$ is a constant. We assume, for brevity, that $\lg(N/f(n))$ and $N/2^L + (1+\varepsilon)\lg L$ are integers.

Block usage. The block usage is the same as the ORAM construction described in Section 3 except that each real block is given *two* position labels instead of one. We call them the *primary position label* and the *secondary position label*. Only the primary position labels are stored in the metadata blocks (as in Section 3).

Data layout. The data layout is basically the same as in Section 3. We only explain the differences from Section 3. First, the position table stores both the primary position labels and the secondary position labels. Second, the user maintains an additional table called *counter table*. It is a size 2^L array whose i-th entry is the number of real blocks with primary position label i. Last, since the value of each of L and M is different from that in Section 3, the tree/bucket size is changed accordingly.

Access procedure. The same invariant conditions as Section 3 are maintained except that the "position label" in the second condition is replaced by "primary position label".

We provide the pseudocode in the full paper. To read the high-level description below, it suffices to know that $P(\ell)$ is the path from the root to the ℓ-th leaf in the data tree.

Let b_a be the accessed block. We first retrieve the two position labels ℓ_1 and ℓ_2 of b_a from the position table and update each of the two position table values to a number chosen independently and uniformly at random from $[L]$, which will become the new position labels of b_a. One of ℓ_1 and ℓ_2 is the primary position label and the other is the secondary position label but we do not know (and do not need to know) which is which. By the invariant conditions, b_a is either in the stash or in $P(\ell_1)$ or $P(\ell_2)$. We scan $P(\ell_1)$ and $P(\ell_2)$ and retrieve b_a from $P(\ell_i)$ if the primary position label is ℓ_i and b_a is in $P(\ell_i)$. If b_a is not found in the paths, it must be in the stash and we retrieve it from the stash. At this point, we know the primary position label ℓ of b_a (since it is written in the pos entry of the block) and we decrement the ℓ-th entry of the counter table, determine the new primary position label ℓ'_i and increment the ℓ'_i-th entry of the counter table. After, that, we update the block contents if it is a write request, call EvictPath and returns the block contents (before update) in the same way as the algorithm in Section 3.

Outsourcing the position/counter table. In the full version of the construction, the position table and the counter table are stored on the server using the Path ORAM. Every read from (resp. write to) each of these tables is done using the Path ORAM access procedure.

4.2 Analysis

Security. The security proof of the current ORAM construction is almost the same as in Section 3. The only difference in the situation is that now, the sequence of accessed addresses $\mathbf{a}'_2$ depends on two position labels instead of one. Anyway, these position labels are distributed independently and uniformly at random and thus, are independent of $\mathbf{a}$.

Server space. The bounds $\log N = \Theta(\log n)$, $L = \Theta(\log n)$ and $M = \Theta(f(n))$ still hold.

The number of blocks in the leaf buckets is $M2^L = N(1 + \frac{(1+\varepsilon)\lg L}{f(n)}) = N(1 + \Theta(\frac{\log\log n}{f(n)}))$. The number of blocks in the internal buckets is $Z(2^L - 1) < ZN/f(n)$, which is $O(\frac{\log\log n}{f(n)})$. Thus, the data tree size is bounded by $N(1+\Theta(\frac{\log\log n}{f(n)}))$ blocks. As in Section 3, the metadata size of each data block is $\Theta(\frac{\log n}{B})$ blocks. Thus, the number of blocks in the data tree and the metadata tree combined is at most $1 + \Theta(\frac{\log n}{B})$ times larger than $N(1 + \Theta(\frac{\log\log n}{f(n)}))$, which is $n(1 + \Theta(\frac{\log n}{B} + \frac{\log\log n}{f(n)}))$ bits.

Position labels take $2NL = 2nL/B \leq 2n\frac{\log n}{B}$ bits while counter table values take $2^L\lceil\lg N\rceil = N\lceil\lg N\rceil/f(n) \leq N = n/B$ bits. By Proposition 2, the Path ORAM containing the position table (resp. counter table) takes $\Theta(n\frac{\log n}{B})$ (resp. $\Theta(n/B)$) bits.

Therefore, the server space usage is bounded by $n(1 + \Theta(\frac{\log n}{B} + \frac{\log\log n}{f(n)}))$ bits.

Bandwidth blowup. By the same argument as in the bandwidth analysis, the bandwidth cost of each of ReadPath and EvictPath is proportional to $ZL + M = O(\log n + f(n))$ (in blocks). By Proposition 2, the bandwidth cost of access to each of the position table and the counter table is $O(\log^2 n)$. Thus, the bandwidth blowup is $O(\log^2 n)$.

Table 2 Performance comparison with concrete parameters. The symbol † means the integration of Ring ORAM techniques. $N = 2^{20}$, $B = 2^{10}$. A and S are parameters for the Ring ORAM. (A specifies the infrequency of EvictPath and S is the space in each bucket reserved for dummy blocks.) The cost for recursive calls and metadata handling are relatively minor and not included. The stash overflow probability is $< 2^{-80}$ for rigorous settings. Aggressive settings do not have security guarantees (stash size bounds) and, in particular, are not suitable for fair comparison.

		Parameters Z, L, M, A, S	Extra server space	Bandwidth	Stash size
Rigorous	[38]	5,20,–,–,–	$9N$	210	114
	[32]	5,19,–,4,6	$10N$	109	63
	Th. 3	3,15,112,–,–	$2.59N$	471	32
	Th. 3†	5,15,112,4,7	$2.91N$	253	64
Aggressive	[38]	4,19,–,–,–	$3N$	160	
	[32]	5,19,–,4,6	$7N$	145	
	Th. 3	4,15,36,–,–	$.25N$	288	
	Th. 3†	5,15,36,4,6	$.46875N$	163	
	Th. 5	3,16,14,–,–	$.0625N$	248	
	Th. 5†	5,15,28,4,7	$.25N$	194	

User space. By the same argument as in the user space analysis in Section 3, the temporary user space is proportional to $ZL + M = O(\log n + f(n))$. We prove the bound on the permanent user space usage, i.e., the

5 Practicality of the Proposed Methods

Table 2 shows the performance of the proposed methods, the Path ORAM [38] and the Ring ORAM [32] with concrete parameters. The Ring ORAM has asymptotically the same performance as the Path ORAM but it achieves constant factor smaller bandwidth at the cost of larger server space. It is easy to integrate the main technique of the Ring ORAM to the internal nodes of the proposed methods and we also show the performance of these variants. We show the integration itself in the full paper.

The table contains "rigorous" and "aggressive" parameter settings. Rigorous parameters were derived from theoretical analysis with additional care for constant factors. The aggressive parameters for existing methods were taken from the experiments in the original papers. We chose the aggressive parameters for the proposed methods by simulation: we simulated database scan (accessing addresses $1, 2, \ldots, N$) for 100 times and found some parameters for which the stash size after every scan was zero. (Such usage of scan is standard in literature since scan maximizes the stash size.) We emphasize that constructions with aggressive parameters lack rigorous security and they are not suitable for fair comparison.

Unfortunately, we could not derive rigorous bounds for the second construction (Theorem 5) for reasonable size of N since the balls-into-bins analysis of Berenbrink et al. [3], used in the stash size analysis, requires a very large number of bins. However, the simulation results indicate that the second construction works for reasonable size of N.

6 Conclusion

ORAM is a multifaceted problem and recently, researchers have been recognizing the importance of rethinking the relevancy of multiple aspects of ORAM using modern standards [37, 4].

In this paper, we provided another point of view and insight for this exploration by introducing the notion of succinctness to ORAM and proposing succinct ORAM constructions. We think our methods are particularly suitable for secure processor setting. It is interesting to consider succinct constructions optimized for other settings.

References

1. Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM J. Comput.*, 29(1):180–200, 1999. doi:10.1137/S0097539795288490.
2. David Benoit, Erik D. Demaine, J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Representing trees of higher degree. *Algorithmica*, 43(4):275–292, 2005. doi:10.1007/s00453-004-1146-6.
3. Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced allocations: the heavily loaded case. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 745–754. ACM, 2000. doi:10.1145/335305.335411.
4. Vincent Bindschaedler, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, and Yan Huang. Practicing oblivious access on cloud storage: the gap, the fallacy, and the new way forward. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 837–849. ACM, 2015. doi:10.1145/2810103.2813649.
5. Kai-Min Chung, Zhenming Liu, and Rafael Pass. Statistically-secure ORAM with $\tilde{o}(\log^2 n)$ overhead. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 62–81. Springer, 2014. doi:10.1007/978-3-662-45608-8_4.
6. David R. Clark and J. Ian Munro. Efficient suffix trees on secondary storage. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '96, pages 383–391, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=313852.314087.
7. Srinivas Devadas, Marten van Dijk, Christopher W. Fletcher, Ling Ren, Elaine Shi, and Daniel Wichs. Onion ORAM: A constant bandwidth blowup oblivious RAM. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 145–174. Springer, 2016. doi:10.1007/978-3-662-49099-0_6.
8. Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Structuring labeled trees for optimal succinctness, and beyond. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 184–196. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.69.
9. Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.
10. Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. Compressed representations of sequences and full-text indexes. *ACM Trans. Algorithms*, 3(2):20, 2007. doi:10.1145/1240233.1240243.
11. Christopher W. Fletcher, Marten van Dijk, and Srinivas Devadas. A secure processor architecture for encrypted computation on untrusted programs. In *Proceedings of the 7th ACM Workshop on Scalable Trusted Computing*, STC '12, pages 3–8, New York, NY, USA, 2012. ACM. doi:10.1145/2382536.2382540.

12 Christopher W. Fletcher, Ling Ren, Albert Kwon, Marten van Dijk, and Srinivas Devadas. Freecursive ORAM: [nearly] free recursion and integrity verification for position-based oblivious RAM. In Özcan Özturk, Kemal Ebcioglu, and Sandhya Dwarkadas, editors, *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '15, Istanbul, Turkey, March 14-18, 2015*, pages 103–116. ACM, 2015. doi:10.1145/2694344.2694353.

13 Craig Gentry, Kenny A. Goldman, Shai Halevi, Charanjit S. Jutla, Mariana Raykova, and Daniel Wichs. Optimizing ORAM and using it efficiently for secure computation. In Emiliano De Cristofaro and Matthew K. Wright, editors, *Privacy Enhancing Technologies - 13th International Symposium, PETS 2013, Bloomington, IN, USA, July 10-12, 2013. Proceedings*, volume 7981 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013. doi:10.1007/978-3-642-39077-7_1.

14 Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 182–194. ACM, 1987. doi:10.1145/28395.28416.

15 Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Rank/select operations on large alphabets: A tool for text indexing. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 368–373, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1109557.1109599.

16 Michael T. Goodrich and Michael Mitzenmacher. Privacy-preserving access of outsourced data via oblivious ram simulation. In *Proceedings of the 38th International Conference on Automata, Languages and Programming - Volume Part II*, ICALP'11, pages 576–587, Berlin, Heidelberg, 2011. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=2027223.2027282.

17 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 841–850, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=644108.644250.

18 Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Space-efficient frameworks for top-k string retrieval. *J. ACM*, 61(2):9:1–9:36, 2014. doi:10.1145/2590774.

19 Guy Jacobson. Space-efficient static trees and graphs. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 549–554. IEEE Computer Society, 1989. doi:10.1109/SFCS.1989.63533.

20 Guy Joseph Jacobson. *Succinct Static Data Structures*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1988.

21 Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung. Ultra-succinct representation of ordered trees. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 575–584, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1283383.1283445.

22 Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. On the (in)security of hash-based oblivious RAM and a new balancing scheme. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 143–156. SIAM, 2012. doi:10.1137/1.9781611973099.13.

23 Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Krste Asanovic, John Kubiatowicz, and Dawn Song. PHANTOM: practical oblivious computation in a secure processor. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM*

SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013, pages 311–324. ACM, 2013. doi:10.1145/2508859.2516692.

24 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press, New York, NY, USA, 2nd edition, 2017.

25 J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. Comput.*, 31(3):762–776, 2001. doi:10.1137/S0097539799364092.

26 J. Ian Munro, Venkatesh Raman, and S. Srinivasa Rao. Space efficient suffix trees. *J. Algorithms*, 39(2):205–222, 2001. doi:10.1006/jagm.2000.1151.

27 Gonzalo Navarro and Yakov Nekrich. Optimal dynamic sequence representations. *SIAM J. Comput.*, 43(5):1781–1806, 2014. doi:10.1137/130908245.

28 Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Trans. Algorithms*, 10(3):16:1–16:39, 2014. doi:10.1145/2601073.

29 Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004. doi:10.1016/j.jalgor.2003.12.002.

30 Benny Pinkas and Tzachy Reinman. Oblivious ram revisited. In *Proceedings of the 30th Annual Conference on Advances in Cryptology*, CRYPTO'10, pages 502–519, Berlin, Heidelberg, 2010. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=1881412.1881447.

31 Rajeev Raman and S. Srinivasa Rao. Succinct dynamic dictionaries and trees. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings*, volume 2719 of *Lecture Notes in Computer Science*, pages 357–368. Springer, 2003. doi:10.1007/3-540-45061-0_30.

32 Ling Ren, Christopher Fletcher, Albert Kwon, Emil Stefanov, Elaine Shi, Marten van Dijk, and Srinivas Devadas. Constants count: Practical improvements to oblivious ram. In *24th USENIX Security Symposium*, pages 415–430, Washington, D.C., 2015. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/ren-ling.

33 Ling Ren, Xiangyao Yu, Christopher W. Fletcher, Marten van Dijk, and Srinivas Devadas. Design space exploration and optimization of path oblivious ram in secure processors. *SIGARCH Comput. Archit. News*, 41(3):571–582, 2013. doi:10.1145/2508148.2485971.

34 Kunihiko Sadakane. Succinct representations of *lcp* information and improvements in the compressed suffix arrays. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 225–232, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=545381.545410.

35 Kunihiko Sadakane and Roberto Grossi. Squeezing succinct data structures into entropy bounds. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 1230–1239, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1109557.1109693.

36 Elaine Shi, T.-H. Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious RAM with o((logn)3) worst-case cost. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 197–214. Springer, 2011. doi:10.1007/978-3-642-25385-0_11.

37 Emil Stefanov, Elaine Shi, and Dawn Xiaodong Song. Towards practical oblivious ram. In *19th Annual Network and Distributed System Security Symposium*, 2012. URL: http://www.internetsociety.org/towards-practical-oblivious-ram.

38 Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 299–310. ACM, 2013. doi:10.1145/2508859.2516660.

Recursion Schemes and the WMSO+U Logic

Paweł Parys
University of Warsaw, Warsaw, Poland
parys@mimuw.edu.pl

Abstract

We study the w eak MSO logic extended by the unbounding quantifier (WMSO+U), expressing the fact that there exist arbitrarily large finite sets satisfying a given property. We prove that it is decidable whether the tree generated by a given higher-order recursion scheme satisfies a given sentence of WMSO+U.

2012 ACM Subject Classification Theory of computation → Rewrite systems, Theory of computation → Logic and verification

Keywords and phrases Higher-order Recursion Schemes, Intersection Types, WMSO+U Logic, Boundedness

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.53

Funding Work supported by the National Science Centre, Poland (grant no. 2016/22/E/ST6/00041).

Acknowledgements We thank Szymon Toruńczyk for a discussion on topics covered by this paper.

1 Introduction

Higher-order recursion schemes (*schemes* in short) are used to faithfully represent the control flow of programs in languages with higher-order functions [16, 22, 28, 24]. This formalism is equivalent via direct translations to simply-typed λY-calculus [36]. Collapsible pushdown systems [20] and ordered tree-pushdown systems [13] are other equivalent formalisms. Schemes cover some other models such as indexed grammars [1] and ordered multi-pushdown automata [8].

In our setting, a scheme is a finite description of an infinite tree. A useful property of schemes is that the *MSO-model-checking problem* for schemes is decidable. This means that given a scheme $\mathcal{G}$ and an MSO sentence φ, it can be algorithmically decided whether the tree generated by $\mathcal{G}$ satisfies φ. This result has several different proofs [28, 20, 25, 34], and also some extensions like global model checking [11], logical reflection [9], effective selection [12], existence of λ-calculus model [35]. When the property of trees is given as an automaton, not as a formula, the model-checking problem can be solved efficiently, in the sense that there exist implementations working in a reasonable running time [24, 23, 10, 32, 27] (most tools cover only a fragment of MSO, though).

Recently, an interest arisen in model-checking trees generated by schemes against properties not expressible in the MSO logic. These are properties expressing boundedness and unboundedness of some quantities. More precisely, it was shown that the *diagonal problem* for schemes is decidable [19, 14, 31]. This problem asks, given a scheme $\mathcal{G}$ and a set of letters A, whether for every $n \in \mathbb{N}$ there exists a path in the tree generated by $\mathcal{G}$ such that every letter from A appears on this path at least n times. This result turns out to be interesting, because it entails other decidability results for recursion schemes, concerning in particular

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 53; pp. 53:1–53:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

computability of the downward closure of recognized languages [38], and the problem of separability by piecewise testable languages [15].

In this paper we show a result of a more general style. Instead of considering a particular property, like in the diagonal problem, we consider a logic capable to express properties talking about boundedness. More precisely, we choose the WMSO+U logic. This logic extends WMSO (a fragment of MSO in which one can quantify only over finite sets) by the unbounding quantifier, U [3]. A formula using this quantifier, $\mathsf{U}X.\,\varphi$, says that φ holds for arbitrarily large finite sets X. The WMSO+U logic was widely considered in the context of infinite words [4] and infinite trees [18, 7, 5].

The goal of this paper is to prove the following theorem.

▶ **Theorem 1.** *It is decidable whether the tree generated by a given scheme satisfies a given* WMSO+U *sentence.*

In our solution, we depend on several earlier results. First, we translate WMSO+U formulae to an equivalent automata model using the notion of logical types (aka. composition method) following a long series of previous work (some selection: [17, 37, 26, 2, 18, 30]). Second, we use the logical-reflection property of schemes [9]. It says that given a scheme $\mathcal{G}$ and an MSO sentence φ one can construct a scheme $\mathcal{G}_\varphi$ generating the same tree as $\mathcal{G}$, where in every node it is additionally written whether φ is satisfied in the subtree starting in this node. Third, from our previous work on the diagonal problem [29, 31], we deduce an analogous property for the diagonal problem, which we call *diagonal reflection* (Theorem 6): given a scheme $\mathcal{G}$ we can construct a scheme $\mathcal{G}_{diag}$ generating the same tree as $\mathcal{G}$, where every node is additionally annotated by the solution of the diagonal problem in the subtree starting in this node. We believe that Theorem 6 is a contribution of independent interest. Finally, we use the fact that schemes can be composed with finite tree transducers transforming the generated trees; this follows directly from the equivalence between schemes and collapsible pushdown systems [20].

We remark that the model-checking problem for the full MSO logic (equipped with quantification over infinite sets) combined with the U quantifier is undecidable already over the infinite word without labels [6], so even more over all fancy trees that can be generated by higher-order recursion schemes. For this reason it is necessary to restrict the quantification to finite sets.

Our paper is structured as follows. In Section 2 we introduce all necessary definitions. In Section 3 we show how to translate WMSO+U sentences to automata. In Section 4 we give a theorem concerning diagonal reflection. Next, in Section 5, we finish the proof of the main theorem. We conclude in Section 6 by listing some possible extensions of our results.

2 Preliminaries

The powerset of a set X is denoted $\mathcal{P}(X)$. For a relation r, we write r^* for the reflexive transitive closure of r. When f is a function, by $f[x \mapsto y]$ we mean the function that maps x to y and every other $z \in \mathrm{dom}(f)$ to $f(z)$.

Infinitary λ-calculus. We consider infinitary, simply-typed λ-calculus. In particular, each λ-term has an associated sort (aka. simple type). The set of *sorts* is constructed from a unique ground sort o using a binary operation $\to$; namely o is a sort, and if α and β are sorts, so is $\alpha{\to}\beta$. By convention, $\to$ associates to the right, that is, $\alpha{\to}\beta{\to}\gamma$ is understood as $\alpha{\to}(\beta{\to}\gamma)$.

While defining λ-terms we assume an infinite set of letters Σ (we use unranked letters; this subsumes the setting of ranked letters), and a set of variables $\mathcal{V} = \{x^\alpha, y^\beta, z^\gamma\}$ containing infinitely many variables of every sort (sort of a variable is written in superscript). *Infinitary λ-terms* (or just *λ-terms*) are defined by coinduction, according to the following rules:

- node constructor—if $a \in \Sigma$, and $K_1^\circ, \dots, K_r^\circ$ are λ-terms, then $(a\langle K_1^\circ, \dots, K_r^\circ\rangle)^\circ$ is a λ-term,
- variable—every variable $x^\alpha \in \mathcal{V}$ is a λ-term,
- application—if $K^{\alpha\to\beta}$ and L^α are λ-terms, then $(K^{\alpha\to\beta}\, L^\alpha)^\beta$ is a λ-term, and
- λ-binder—if K^β is a λ-term and x^α is a variable, then $(\lambda x^\alpha.K^\beta)^{\alpha\to\beta}$ is a λ-term.

We naturally identify λ-terms differing only in names of bound variables. We often omit the sort annotations of λ-terms, but we keep in mind that every λ-term (and every variable) has a fixed sort. Free variables and subterms of a λ-term, as well as β-reductions, are defined as usually. A λ-term K is *closed* if it has no free variables. We restrict ourselves to those λ-terms for which the set of sorts of all subterms is finite.

Trees; Böhm Trees. A *tree* is defined as a λ-term that is built using only node constructors, that is, not using variables, applications, nor λ-binders. For a tree $T = a\langle T_1, \dots, T_r\rangle$, its set of nodes is defined as the smallest set such that

- ε is a node of T, labeled by a, and
- if v is a node of T_i for some $i \in \{1, \dots, r\}$, labeled by b, then iv is a node of T, also labeled by b.

A node v is the *i-th child* of u if $v = ui$. We say that two trees T, T' are of *the same shape* if they have the same nodes. By $T{\restriction_v}$ we denote the subtree of T starting in the node v, defined as one expects. For a (usually finite) subset Σ_0 of Σ, and for $r_{\max} \in \mathbb{N}$, a $(\Sigma_0, r_{\max})$-*tree* is a tree in which all node labels belong to Σ_0, and in which every node has at most $r_{\max}$ children.

We consider Böhm trees only for closed λ-terms of sort o. For such a λ-term K, its *Böhm tree* is constructed by coinduction, as follows: if there is a sequence of β-reductions from K to a λ-term of the form $a\langle K_1, \dots, K_r\rangle$, and $T_1, \dots, T_r$ are Böhm trees of $K_1, \dots, K_r$, respectively, then $a\langle T_1, \dots, T_r\rangle$ is a Böhm tree of K; if there is no such sequence of β-reductions from K, then $\omega\langle\rangle$ is a Böhm tree of K (where $\omega \in \Sigma$ is a fixed letter). It is folklore that every closed λ-term of sort o has exactly one Böhm tree (the order in which β-reductions are performed does not matter); this tree is denoted by $BT(K)$.

A closed λ-term K of sort o is called *fully convergent* if every node of $BT(K)$ is explicitly created by a node constructor from K (e.g., $\omega\langle\rangle$ is fully convergent, while $K = (\lambda x^\circ.x)\, K$ is not). More formally: we consider the λ-term $K_{-\omega}$ obtained from K by replacing ω with some other letter ω', and we say that K is fully convergent if in $BT(K_{-\omega})$ there are no ω-labeled nodes.

Higher-Order Recursion Schemes. Our definition of schemes is less restrictive than usually, as we see them only as finite representations of infinite λ-terms. Thus a *higher-order recursion scheme* (or just a *scheme*) is a triple $\mathcal{G} = (\mathcal{N}, \mathcal{R}, N_0^\circ)$, where $\mathcal{N} \subseteq \mathcal{V}$ is a finite set of nonterminals, $\mathcal{R}$ is a function that maps every nonterminal $N \in \mathcal{N}$ to a finite λ-term whose all free variables are contained in $\mathcal{N}$ and whose sort equals the sort of N, and $N_0^\circ \in \mathcal{N}$ is a starting nonterminal, being of sort o. We assume that elements of $\mathcal{N}$ are not used as bound variables, and that $\mathcal{R}(N)$ is not a nonterminal for any $N \in \mathcal{N}$.

For a scheme $\mathcal{G} = (\mathcal{N}, \mathcal{R}, N_0)$, and for a λ-term K whose free variables are contained in $\mathcal{N}$, we define the infinitary λ-term *generated by* $\mathcal{G}$ from K, denoted $\Lambda_{\mathcal{G}}(K)$, by coinduction:

to obtain $\Lambda_{\mathcal{G}}(K)$ we replace in K every nonterminal $N \in \mathcal{N}$ with $\Lambda_{\mathcal{G}}(\mathcal{R}(N))$. Observe that $\Lambda_{\mathcal{G}}(K)$ is a closed λ-term of the same sort as K. The infinitary λ-term *generated by* $\mathcal{G}$, denoted $\Lambda(\mathcal{G})$, equals $\Lambda_{\mathcal{G}}(N_0)$.

By the *tree generated by* $\mathcal{G}$ we mean $BT(\Lambda(\mathcal{G}))$. We write $\Sigma_{\mathcal{G}}$ for the finite subset of Σ containing ω and letters used in node constructors appearing in $\mathcal{G}$, and $r_{\max}(\mathcal{G})$ for the maximal arity of node constructors appearing in $\mathcal{G}$. Clearly $BT(\Lambda(\mathcal{G}))$ is a $(\Sigma_{\mathcal{G}}, r_{\max}(\mathcal{G}))$-tree.

In our constructions it is convenient to consider only schemes generating fully-convergent λ-terms, which is possible due to the following standard result.

▶ **Fact 2** ([33, page 14]). *For every scheme $\mathcal{G}$ we can construct a scheme $\mathcal{G}'$ generating the same tree as $\mathcal{G}$, and such that $\Lambda(\mathcal{G}')$ is fully convergent.*

▶ **Example.** *Consider the scheme* $\mathcal{G}_1 = (\{\mathsf{M}^{\circ}, \mathsf{N}^{\circ\to\circ}\}, \mathcal{R}, \mathsf{M})$, *where*

$$\mathcal{R}(\mathsf{N}) = \lambda x^{\circ}.a\langle x, \mathsf{N}\,(b\langle x\rangle)\rangle\,, \qquad \text{and} \qquad \mathcal{R}(\mathsf{M}) = \mathsf{N}\,(c\langle\rangle)\,.$$

We obtain $\Lambda(\mathcal{G}_1) = K\,(c\langle\rangle)$, *where K is the unique λ-term such that* $K = \lambda x^{\circ}.a\langle x, K\,(b\langle x\rangle)\rangle$. *The tree generated by* $\mathcal{G}_1$ *equals* $a\langle T_0, a\langle T_1, a\langle T_2, \ldots\rangle\rangle\rangle$, *where* $T_0 = c\langle\rangle$ *and* $T_i = b\langle T_{i-1}\rangle$ *for all* $i \geq 1$.

WMSO+U. For technical convenience, we use a variant of WMSO+U in which there are no first-order variables. It is easy to translate a formula from any standard syntax of WMSO+U to ours (at least when the maximal arity of considered trees is fixed). In the syntax of WMSO+U we have the following constructions:

$$\varphi ::= a(X) \mid X \swarrow_i Y \mid X \subseteq Y \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi' \mid \exists_{\mathsf{fin}} X.\varphi' \mid \mathsf{U}X.\varphi' \qquad \text{where } a \in \Sigma, i \in \mathbb{N}_+.$$

We evaluate formulae of WMSO+U in Σ-labeled trees. Set variables are interpreted as finite sets of nodes, and the semantics of formulae is defined as follows:

- $a(X)$ holds when every node in X is labeled by a,
- $X \swarrow_i Y$ holds when both X and Y are singletons, and the unique node in Y is the i-th child of the unique node in X,
- $X \subseteq Y$, $\varphi_1 \wedge \varphi_2$, and $\neg\varphi'$ are defined as expected,
- $\exists_{\mathsf{fin}} X.\varphi'$ holds when φ' holds for some finite set of nodes X, and
- $\mathsf{U}X.\varphi'$ holds when for every $n \in \mathbb{N}$, φ' holds for some finite set of nodes X of cardinality at least n.

3 Nested U-Prefix Automata

In this section we give a definition of nested U-prefix automata, a formalism equivalent to the WMSO+U logic. A U-*prefix automaton* is a pair $\mathcal{A} = (Q, Q_{\mathsf{imp}}, \Delta)$, where Q is a finite set of states, $Q_{\mathsf{imp}} \subseteq Q$ is a set of *important* states, and $\Delta \subseteq Q \times \Sigma \times (Q \cup \{\top\})^*$ is a finite transition relation (we assume $\top \notin Q$). A *run* of $\mathcal{A}$ on a tree T is a mapping ρ from the set of nodes of T to $Q \cup \{\top\}$ such that

- there are only finitely many nodes v such that $\rho(v) \in Q$, and
- for every node v of T, with label a and r children, it holds that either $\rho(v) = \top = \rho(v1) = \cdots = \rho(vr)$ or $(\rho(v), a, \rho(v1), \ldots, \rho(vr)) \in \Delta$.

We use U-prefix automata as transducers, relabeling nodes of T: we define $\mathcal{A}(T)$ to be the tree of the same shape as T, and such that its every node v is labeled by a function $f_v \colon Q \to \{0, 1, 2\}$, which assigns to every state $q \in Q$:

- 2, if for every $n \in \mathbb{N}$ there is a run ρ_n of $\mathcal{A}$ on $T{\restriction_v}$ that assigns q to the root of $T{\restriction_v}$, and such that for at least n nodes w it holds that $\rho_n(w) \in Q_{\mathsf{imp}}$;
- 1, if the above does not hold, but there is a run of $\mathcal{A}$ on $T{\restriction_v}$ that assigns q to the root of $T{\restriction_v}$;
- 0, if none of the above holds.

By the *output alphabet* of $\mathcal{A}$ we mean the set of functions $\Sigma^{\mathsf{out}}(\mathcal{A}) = \{0,1,2\}^Q$; we assume that $\{0,1,2\}^Q \subseteq \Sigma$.

A *nested* U*-prefix automaton* is a sequence $\mathcal{A} = \mathcal{A}_1 \circ \cdots \circ \mathcal{A}_k$ of U-prefix automata, where $k \geq 1$. We define $\mathcal{A}(T)$ to be $\mathcal{A}_k(\ldots(\mathcal{A}_1(T))\ldots)$. The output alphabet of $\mathcal{A}$, denoted $\Sigma^{\mathsf{out}}(\mathcal{A})$, equals $\Sigma^{\mathsf{out}}(\mathcal{A}_k)$. The key property is that these automata can check properties expressed in WMSO+U (actually, they are equivalent to WMSO+U, but we need only the one direction here).

▶ **Lemma 3.** *Let $\Sigma_0 \subseteq \Sigma$ be a finite fragment of the alphabet, and let $r_{\max} \in \mathbb{N}$. Then for every* WMSO+U *sentence φ we can construct a nested U-prefix automaton $\mathcal{A}$, and a subset $\Sigma_{\mathsf{F}} \subseteq \Sigma^{\mathsf{out}}(\mathcal{A})$ such that for every $(\Sigma_0, r_{\max})$-tree T, it holds that T satisfies φ if and only if the root of $\mathcal{A}(T)$ is labeled by a letter in Σ_{F}.*

We remark that Bojańczyk and Toruńczyk [7] introduce another model of automata equivalent to WMSO+U: nested limsup automata. A common property of these two models is that both of them are nested, but the components are of different form.

Recall that our aim is to evaluate φ in a tree T generated by a particular recursion scheme $\mathcal{G}$, so the restriction to $(\Sigma_0, r_{\max})$-trees is not harmful: as $(\Sigma_0, r_{\max})$ we are going to take $(\Sigma_{\mathcal{G}}, r_{\max}(\mathcal{G}))$.

We now come to the proof of Lemma 3. We notice that due to the nested structure, our automata are quite close to the logic. Nondeterminism on particular levels of the automaton may realize the choices done by particular quantifiers of the formula. Moreover, in effect of applying an automaton we check whether something is unbounded, which corresponds to the U quantifiers. As states of the automaton we will take *phenotypes* (aka. logical types), which are defined next.

Fix some finite set $\mathcal{F}$ of variables, such that all variables appearing in WMSO+U formulae under consideration come from this set. Let φ be a formula of WMSO+U, let T be a tree, and let ν be a valuation assigning finite sets of nodes of T to variables from $\mathcal{F}$. We define the φ*-phenotype* of T under valuation ν, denoted $[T]^\nu_\varphi$, by induction on the size of φ as follows:

- if φ is of the form $a(X)$ (for some symbol $a \in \Sigma$) or $X \subseteq Y$ then $[T]^\nu_\varphi$ is the logical value of φ in T, ν, that is, tt if $T, \nu \models \varphi$ and ff otherwise,
- if φ is of the form $X \not\perp_i Y$, then $[T]^\nu_\varphi$ equals:
  - tt if $T, \nu \models \varphi$,
  - empty if $\nu(X) = \nu(Y) = \emptyset$,
  - root if $\nu(X) = \emptyset$ and $\nu(Y) = \{\varepsilon\}$, and
  - ff otherwise,
- if $\varphi = (\psi_1 \wedge \psi_2)$, then $[T]^\nu_\varphi = ([T]^\nu_{\psi_1}, [T]^\nu_{\psi_2})$,
- if $\varphi = (\neg\psi)$, then $[T]^\nu_\varphi = [T]^\nu_\psi$, and
- if $\varphi = \exists_{\mathsf{fin}} X.\psi$ or $\varphi = \mathsf{U}X.\psi$, then

$$[T]^\nu_\varphi = \left(\{\sigma \mid \exists X^T.\, [T]^{\nu[X\mapsto X^T]}_\psi = \sigma\}, \{\sigma \mid \forall n.\, \exists X^T.\, [T]^{\nu[X\mapsto X^T]}_\psi = \sigma \wedge |X^T| \geq n\}\right),$$

where X^T ranges over finite sets of nodes of T and n ranges over $\mathbb{N}$.

For each φ, let Pht_φ denote the set of all potential φ-phenotypes. Namely, $Pht_\varphi = \{\mathsf{tt}, \mathsf{ff}\}$ in the first case, $Pht_\varphi = \{\mathsf{tt}, \mathsf{empty}, \mathsf{root}, \mathsf{ff}\}$ in the second case, $Pht_\varphi = Pht_{\psi_1} \times Pht_{\psi_2}$ in the third case, $Pht_\varphi = Pht_\psi$ in the fourth case, and $Pht_\varphi = (\mathcal{P}(Pht_\psi))^2$ in the last case.

We immediately see two facts. First, Pht_φ is finite for every φ. Second, the fact whether φ holds in T, ν is determined by $[T]^\nu_\varphi$. This means that there is a function $tv_\varphi \colon Pht_\varphi \to \{\mathsf{tt}, \mathsf{ff}\}$ such that $tv_\varphi([T]^\nu_\varphi) = \mathsf{tt}$ if and only if $T, \nu \models \varphi$.

Next, we observe that phenotypes behave in a compositional way, as formalized below. Here for a valuation ν and a node v, by $\nu{\restriction_v}$ we mean the valuation that restricts ν to the subtree starting at v, that is, maps every variable $X \in \mathcal{F}$ to $\{w \mid vw \in \nu(X)\}$.

▶ **Lemma 4** (cf. [18, 30]). *For every letter $a \in \Sigma$, every $r \in \mathbb{N}$, and every formula φ, one can compute a function $Comp_{a,r,\varphi} \colon \mathcal{P}(\mathcal{F}) \times (Pht_\varphi)^r \to Pht_\varphi$ such that for every tree T whose root has label a and r children, and for every valuation ν,*

$$[T]^\nu_\varphi = Comp_{a,r,\varphi}(\{X \in \mathcal{F} \mid \varepsilon \in \nu(X)\}, [T{\restriction_1}]^{\nu{\restriction_1}}_\varphi, \ldots, [T{\restriction_r}]^{\nu{\restriction_r}}_\varphi)\,.$$

Proof. We proceed by induction on the size of φ.

When φ is of the form $b(X)$ or $X \subseteq Y$, then we see that φ holds in T, ν if and only if it holds in every subtree $T{\restriction_i}, \nu{\restriction_i}$ and in the root of T. Thus for $\varphi = b(X)$ as $Comp_{a,r,\varphi}(R, \tau_1, \ldots, \tau_r)$ we take tt when $\tau_i = \mathsf{tt}$ for all $i \in \{1, \ldots, r\}$ and either $a = b$ or $X \notin R$. For $\varphi = (X \subseteq Y)$ the last part of the condition is replaced by "if $X \in R$ then $Y \in R$".

Next, suppose that $\varphi = (X \not\downarrow_k Y)$. Then as $Comp_{a,r,\varphi}(R, \tau_1, \ldots, \tau_r)$ we take

- tt if $\tau_j = \mathsf{tt}$ for some $j \in \{1, \ldots, r\}$, and $\tau_i = \mathsf{empty}$ for all $i \in \{1, \ldots, r\} \setminus \{j\}$, and $X \notin R$, and $Y \notin R$,
- tt also if $\tau_k = \mathsf{root}$, and $\tau_i = \mathsf{empty}$ for all $i \in \{1, \ldots, r\} \setminus \{k\}$, and $X \in R$, and $Y \notin R$,
- empty if $\tau_i = \mathsf{empty}$ for all $i \in \{1, \ldots, r\}$, and $X \notin R$, and $Y \notin R$,
- root if $\tau_i = \mathsf{empty}$ for all $i \in \{1, \ldots, r\}$, and $X \notin R$, and $Y \in R$, and
- ff otherwise.

By comparing this definition with the definition of the phenotype we immediately see that the thesis is satisfied.

When $\varphi = (\neg\psi)$, we simply take $Comp_{a,r,\varphi} = Comp_{a,r,\psi}$, and when $\varphi = (\psi_1 \wedge \psi_2)$, as $Comp_{a,r,\varphi}(R, (\tau^1_1, \tau^2_1), \ldots, (\tau^1_r, \tau^2_r))$ we take the pair of $Comp_{a,r,\psi_i}(R, \tau^i_1, \ldots, \tau^i_r)$ for $i \in \{1, 2\}$.

Finally, suppose that $\varphi = \exists_{\mathsf{fin}} X.\psi$ or $\varphi = \mathsf{U}X.\psi$. Let A be the set of tuples $(\sigma_1, \ldots, \sigma_r) \in \tau_1 \times \cdots \times \tau_r$, and let B be the set of tuples $(\sigma_1, \ldots, \sigma_r)$ such that $\sigma_j \in \rho_j$ for some $j \in \{1, \ldots, r\}$ and $\sigma_i \in \tau_i$ for all $i \in \{1, \ldots, r\} \setminus \{j\}$. As $Comp_{a,r,\varphi}(R, (\tau_1, \rho_1), \ldots, (\tau_r, \rho_r))$ we take

$$\begin{aligned}&(\{Comp_{a,r,\psi}(R \cup \{X\}, \sigma_1, \ldots, \sigma_r), Comp_{a,r,\psi}(R \setminus \{X\}, \sigma_1, \ldots, \sigma_r) \mid (\sigma_1, \ldots, \sigma_r) \in A\},\\ &\{Comp_{a,r,\psi}(R \cup \{X\}, \sigma_1, \ldots, \sigma_r), Comp_{a,r,\psi}(R \setminus \{X\}, \sigma_1, \ldots, \sigma_r) \mid (\sigma_1, \ldots, \sigma_r) \in B\})\,.\end{aligned}$$

The two possibilities, $R \cup \{X\}$ and $R \setminus \{X\}$, correspond to the fact that when quantifying over X, the root of T may be either taken to X or not. The second coordinate is computed correctly due to the pigeonhole principle: if for every n we have a set X^T_n of cardinality at least n (satisfying some property), then we can choose an infinite subsequence of these sets such that either the root belongs to all of them or to none of them, and one can choose some $j \in \{1, \ldots, r\}$ such that the sets contain unboundedly many descendants of j. ◀

We now concentrate on phenotypes under the valuation $\nu_\emptyset$ that maps every variable to the empty set.

▶ **Lemma 5.** *Let $\Sigma_0 \subseteq \Sigma$ be a finite fragment of the alphabet, and let $r_{\max} \in \mathbb{N}$. Then for every* WMSO+U *formula φ we can construct a nested* U*-prefix automaton $\mathcal{A}$, and a function $f \colon \Sigma^{\mathsf{out}}(\mathcal{A}) \to Pht_\varphi$ such that for every $(\Sigma_0, r_{\max})$-tree T the root of $\mathcal{A}(T)$ is labeled by a letter η such that $f(\eta) = [T]^{\nu_\emptyset}_\varphi$.*

Proof. Induction on the size of φ. Since all variables are mapped to the empty set, if φ is of the form $a(X)$ or $X \subseteq Y$, then the φ-phenotype of every tree is tt. Thus every $\mathcal{A}$ works fine, only f has to map whole its output alphabet to tt. Similarly, if $\varphi = (X \not\perp_i Y)$, the φ-phenotype is always empty. For $\varphi = (\neg\psi)$ the situation is also trivial: we can directly use the induction assumption since $[T]^{\nu_\emptyset}_{\varphi} = [T]^{\nu_\emptyset}_{\psi}$.

Suppose that $\varphi = (\psi_1 \wedge \psi_2)$. Then from the induction assumption we have automata $\mathcal{B}$ and $\mathcal{C}$ (together with functions g and h) computing ψ_1-phenotypes and ψ_2-phenotypes. It is a routine to alter $\mathcal{B}$ so that from the label of every node v in the output tree $\mathcal{B}(T)$ one can read the original label of v from T (this amounts to adding Σ_0 to the state set of every layer, together with appropriate transitions). We also alter $\mathcal{C}$ so that it reads the output alphabet of $\mathcal{B}$ instead Σ_0; it bases its operation on the original labels from T that can be recovered from the letters, and it copies the information about ψ_1-phenotypes, so that it can be read at the end. After these modifications, we take $\mathcal{A} = \mathcal{B} \circ \mathcal{C}$. Then from the label of the root of $\mathcal{A}(T)$ one can read both $[T]^{\nu_\emptyset}_{\psi_1}$ (copied from the output of $\mathcal{B}$) and $[T]^{\nu_\emptyset}_{\psi_2}$ (calculated by $\mathcal{C}$), so $[T]^{\nu_\emptyset}_{\varphi}$ can be determined.

Finally, suppose that $\varphi = \exists_{\mathsf{fin}} X.\psi$ or $\varphi = \mathsf{U}X.\psi$. By the induction assumption we have an automaton $\mathcal{B}$ and a function g such that for every node v of T, the root of $\mathcal{B}(T{\restriction_v})$ is labeled by a letter η_v such that $g(\eta_v) = [T{\restriction_v}]^{\nu_\emptyset}_{\psi}$. As before, we can also assume that there is a function h such that additionally $h(\eta_v)$ is the original label of v in T. Recall that $\mathcal{B}(T)$ has the same shape as T, and actually $(\mathcal{B}(T)){\restriction_v} = \mathcal{B}(T{\restriction_v})$ for every node v. We construct a new layer $\mathcal{A}'$, which calculates φ-phenotypes basing on ψ-phenotypes, and we take $\mathcal{A} = \mathcal{B} \circ \mathcal{A}'$. As the state set of $\mathcal{A}'$ we take $Q = \{0,1\} \times \mathit{Pht}_\psi$; states from $\{1\} \times \mathit{Pht}_\psi$ are considered as important. Transitions are determined by the *Comp* predicate from Lemma 4. More precisely, for every $r \leq r_{\max}$, every $\eta \in \Sigma^{\mathsf{out}}(\mathcal{B})$, and all $((i_1,\sigma_1),\dots,(i_r,\sigma_r)) \in Q^r$ we have transitions

$$((0, \mathit{Comp}_{h(\eta),r,\psi}(\emptyset,\sigma_1,\dots,\sigma_r)),\eta,(i_1,\sigma_1),\dots,(i_r,\sigma_r))\,, \qquad \text{and}$$
$$((1, \mathit{Comp}_{h(\eta),r,\psi}(\{X\},\sigma_1,\dots,\sigma_r)),\eta,(i_1,\sigma_1),\dots,(i_r,\sigma_r))\,.$$

Moreover, we have transitions that read the ψ-phenotype from the label:

$$((0,g(\eta)),\eta,\underbrace{\top,\dots,\top}_{r}) \qquad \text{for } r \leq r_{\max}.$$

We notice that there is a direct correspondence between runs of $\mathcal{A}'$ and choices of a set of nodes X^T to which the variable X is mapped. The first coordinate of the state is set to 1 in nodes chosen to the set X^T. The second coordinate contains the ψ-phenotype under the valuation mapping X to X^T and every other variable to the empty set. In some nodes below the chosen set X^T we use the transitions of the second kind, reading the ψ-phenotype from the label; it does not matter in which nodes this is done, as everywhere a correct ψ-phenotype is written. The fact that we quantify only over finite sets X^T corresponds to the fact that the run of $\mathcal{A}'$ can assign non-$\top$ states only to a finite prefix of the tree. Moreover, the cardinality of X^T is reflected by the number of important states assigned by a run. It follows that for every $\sigma \in \mathit{Pht}_\psi$,

- there exists a finite set X^T of nodes of T such that $[T]^{\nu_\emptyset[X\mapsto X^T]}_{\psi} = \sigma$ if and only if for some $i \in \{0,1\}$ there is a run of $\mathcal{A}'$ on $\mathcal{B}(T)$ that assigns (i,σ) to the root, and
- for every $n \in \mathbb{N}$ there exists a finite set X^T_n of nodes of T such that $[T]^{\nu_\emptyset[X\mapsto X^T_n]}_{\psi} = \sigma$ and $|X^T_n| \geq n$ if and only if for some $i \in \{0,1\}$ and for every $n \in \mathbb{N}$ there is a run ρ_n of $\mathcal{A}'$ on $\mathcal{B}(T)$ that assigns (i,σ) to the root, and such that ρ_n assigns an important state to at least n nodes.

Thus looking at the root's label in $\mathcal{A}(T)$ we can determine $[T]^{\nu_\emptyset}_{\varphi}$. ◀

Now the proof of Lemma 3 follows easily. Indeed, when φ is a sentence (has no free variables), $[T]^{\nu_\emptyset}_\varphi$ determines whether φ holds in T. Thus it is enough to take the automaton $\mathcal{A}$ constructed in Lemma 5, and replace the function f by the set $\Sigma_{\mathsf{F}} = \{\eta \in \Sigma^{\mathsf{out}}(\mathcal{A}) \mid tv_\varphi(f(\eta))\}$.

4 Diagonal Reflection

The goal of this section is to justify the property of diagonal reflection (Theorem 6).

By $\#_a(U)$ we denote the number of a-labeled nodes in a (finite) tree U. For a set of (finite) trees $\mathcal{L}$ and a set of symbols A, we define a predicate $\mathsf{Diag}_A(\mathcal{L})$, which holds if for every $n \in \mathbb{N}$ there is some $U_n \in \mathcal{L}$ such that for all $a \in A$ it holds that $\#_a(U_n) \geq n$.

Originally, in the diagonal problem we consider nondeterministic higher-order recursion schemes, which instead of generating a single infinite tree, recognize a set of finite trees. We use here an equivalent formulation, in which the set of finite trees is encoded in a single infinite tree. To this end, we use a special letter $\mathsf{nd} \in \Sigma$, denoting a nondeterministic choice. We write $T \to_{\mathsf{nd}} U$ if U is obtained from T by choosing some nd-labeled node u not having any nd-labeled ancestors, and some its child v, and attaching $T{\restriction}_v$ in place of $T{\restriction}_u$. In other words, $\to_{\mathsf{nd}}$ is the smallest relation such that $\mathsf{nd}\langle T_1, \ldots, T_r\rangle \to_{\mathsf{nd}} T_j$ for $j \in \{1, \ldots, r\}$, and if $T_j \to_{\mathsf{nd}} T'_j$ for some $j \in \{1, \ldots, r\}$, and $T_i = T'_i$ for all $i \in \{1, \ldots, r\} \setminus \{j\}$, then $a\langle T_1, \ldots, T_r\rangle \to_{\mathsf{nd}} a\langle T'_1, \ldots, T'_r\rangle$. For a tree T, $\mathcal{L}(T)$ is the set of all finite trees U such that $\#_{\mathsf{nd}}(U) = 0$ and $T \to^*_{\mathsf{nd}} U$.

▶ **Theorem 6** (diagonal reflection). *For every scheme $\mathcal{G}$ generating a tree T one can construct a scheme $\mathcal{G}_{diag}$ that generates a tree of the same shape as T, and such that its every node v is labeled by a pair $(a, \mathcal{D})$, where a is the label of v in T, and $\mathcal{D} = \{A \subseteq \Sigma_{\mathcal{G}} \mid \mathsf{Diag}_A(\mathcal{L}(T{\restriction}_v))\}$.*

While proving this theorem, we depend on our previous work on the diagonal problem [31]. We have developed there a type system, in which for a closed λ-term K we derive type judgments of the form $\vdash_{m,A} K : \hat{\tau} \triangleright c$, where

- m is a natural number,
- $A \subseteq \Sigma$ is the set of types for which we want to solve the diagonal problem (originally it was not written in the type judgment, but anyway the type system depends on this set),
- $\hat{\tau}$ comes from a finite set $\mathcal{TT}^{\alpha}_{m,A}$, depending on m, on A, and on the sort α of K,
- c is a function from A to $\mathbb{N}$.

We refer to type judgments only for closed λ-term, but we remark that they were defined also for λ-terms with free variables (and then one writes a type environment to the left of $\vdash$). While working with some scheme $\mathcal{G}$, as K we only take λ-terms in which all variables have the same sort as some variables appearing in $\Lambda(\mathcal{G})$. Under this assumption, it is enough to consider as m only one fixed value, denoted $m_{\mathcal{G}}$ (equal to the so-called order of $\mathcal{G}$).

Having in mind some scheme $\mathcal{G}$, we define the *value* of a closed λ-term K, denoted $[\![K]\!]$, as the pair consisting of:

- the set of pairs $(A, \hat{\tau})$ such that $A \subseteq \Sigma_{\mathcal{G}}$ and there exists $c\colon A \to \mathbb{N}$ for which $\vdash_{m_{\mathcal{G}},A} K : \hat{\tau} \triangleright c$ can be derived, and
- the set of pairs $(A, \hat{\tau})$ such that $A \subseteq \Sigma_{\mathcal{G}}$ and for every $n \in \mathbb{N}$ there exists $c_n\colon A \to \mathbb{N}$ satisfying $c_n(a) \geq n$ for all $a \in A$, and for which $\vdash_{m_{\mathcal{G}},A} K : \hat{\tau} \triangleright c_n$ can be derived.

When K is of sort α, $[\![K]\!]$ belongs to the finite set $\mathcal{S}^\alpha = (\mathcal{P}(\bigcup_{A \subseteq \Sigma_{\mathcal{G}}} \{A\} \times \mathcal{TT}^{\alpha}_{m_{\mathcal{G}},A}))^2$.

The considered type system is compositional, in the sense that knowing what can be derived for closed λ-terms $K^{\alpha\to\beta}$ and L^α, we can determine what can be derived for $K\,L$. In other words, we can define a composition operation "$\cdot$" on values, going from $\mathcal{S}^{\alpha\to\beta} \times \mathcal{S}^\alpha$ to $\mathcal{S}^\beta$ and such that $[\![K\,L]\!] = [\![K]\!] \cdot [\![L]\!]$ for every closed λ-term $K\,L$.

We now extend the definition of the value to λ-terms K that are not closed. To this end, we need a valuation ν mapping some variables x^α to elements of $\mathcal{S}^\alpha$, which is defined at least for all free variables of K. Then the *value* of K (with respect to ν), denoted $[\![K]\!]^\nu$, is defined as $[\![\lambda x_1.\cdots.\lambda x_k.K]\!]\cdot\nu(x_1)\cdot\ldots\cdot\nu(x_k)$, where $x_1,\ldots,x_k$ are free variables of K, listed according to some fixed order.

Using an algorithm from [31] we can compute $[\![\lambda x_1.\cdots.\lambda x_k.K]\!]$ for every subterm K of $\Lambda(\mathcal{G})$, where, as above, $x_1,\ldots,x_k$ are free variables of K (recall that $\Lambda(\mathcal{G})$ has finitely many subterms).

Having the above properties in hand, it is easy to deduce the following lemma.

▶ **Lemma 7.** *For every scheme $\mathcal{G}=(\mathcal{N},\mathcal{R},N_0)$ generating a tree T one can construct a scheme $\mathcal{G}'$ that generates a tree of the same shape as T, and such that its every node v is labeled by a pair $(a,[\![K]\!])$, where K is some λ-term (closed, of sort o) such that $BT(K)=T{\restriction_v}$.*

Proof. This lemma is proven by literally repeating the construction of Salvati and Walukiewicz [33, Section 5]. We recall it here for completeness. Without loss of generality we assume that $\Lambda(\mathcal{G})$ is fully convergent (cf. Fact 2).

For every sort α, let $[\alpha]=\underbrace{\mathsf{o}\to\ldots\to\mathsf{o}\to}_{|\mathcal{S}^\alpha|}\mathsf{o}$. When $\tau_1,\ldots,\tau_{|\mathcal{S}^\alpha|}$ are all elements of $\mathcal{S}^\alpha$, listed in some fixed order, we let $(\tau_i)_\lambda=\lambda x_1^{\mathsf{o}}.\cdots.\lambda x_{|\mathcal{S}^\alpha|}^{\mathsf{o}}.x_i$ for $i\in\{1,\ldots,|\mathcal{S}^\alpha|\}$; these λ-terms are of sort $[\alpha]$. Given a λ-term K of sort $[\alpha]$, and $K_1,\ldots,K_{|\mathcal{S}^\alpha|}$ of sort $\beta_1\to\ldots\to\beta_s\to\mathsf{o}$, we write $\mathtt{case}\,K\,\{\tau_i\rightsquigarrow K_i\}_{\tau_i\in\mathcal{S}^\alpha}$ for

$$\lambda y_1^{\beta_1}.\cdots.\lambda y_s^{\beta_s}.K\,(K_1\,y_1\,\ldots\,y_s)\,\ldots\,(K_{|\mathcal{S}^\alpha|}\,y_1\,\ldots\,y_s)\,.$$

We notice that for $K=(\tau_j)_\lambda$ this λ-term β-reduces to $\lambda y_1^{\beta_1}.\cdots.\lambda y_s^{\beta_s}.K_j\,y_1\,\ldots\,y_s$, which in turn is η-equivalent to K_j.

We transform every finite λ-term K of sort α to a λ-term $(\!|K|\!)^\nu$ of sort $\alpha^\bullet$, where sorts $\alpha^\bullet$ are defined by induction: $(\alpha\to\beta)^\bullet=\alpha^\bullet\to[\alpha]\to\beta^\bullet$ and $\mathsf{o}^\bullet=\mathsf{o}$. The translation is defined as follows:

$$\begin{aligned}(\!|a\langle K_1,\ldots,K_r\rangle|\!)^\nu&=(a,[\![\Lambda_{\mathcal{G}}(a\langle K_1,\ldots,K_r\rangle)]\!]^\nu)\langle(\!|K_1|\!)^\nu,\ldots,(\!|K_r|\!)^\nu\rangle\,,\\ (\!|x^\alpha|\!)^\nu&=x^{\alpha^\bullet}\,,\\ (\!|K\,L|\!)^\nu&=(\!|K|\!)^\nu\,(\!|L|\!)^\nu\,([\![\Lambda_{\mathcal{G}}(L)]\!]^\nu)_\lambda\,,\\ (\!|\lambda x^\alpha.K|\!)^\nu&=\lambda x^{\alpha^\bullet}.\lambda y^{[\alpha]}.\mathtt{case}\,y\,\{\tau\rightsquigarrow(\!|K|\!)^{\nu[x^\alpha\mapsto\tau]}\}_{\tau\in\mathcal{S}^\alpha}\,.\end{aligned}$$

In the above translation nonterminals are treated as any other variables.

To the resulting scheme $\mathcal{G}'$ we take a nonterminal $N^{\alpha^\bullet}$ for every nonterminal N^α of $\mathcal{G}$, and we define $\mathcal{R}'(N^{\alpha^\bullet})=(\!|\mathcal{R}(N^\alpha)|\!)^\emptyset$, where $\emptyset$ is the valuation with empty domain. It is not difficult to see that such a scheme $\mathcal{G}'$ has the expected properties. We remark that when in effect of performing β-reductions one obtains a λ-term $K=(a,\tau)\langle K_1,\ldots,K_r\rangle$, then $\tau=[\![L]\!]$ for some λ-term L β-equivalent to K, but not necessarily for $L=K$ (it is not clear from [31] whether for β-equivalent λ-terms K and L it holds that $[\![K]\!]=[\![L]\!]$). This is enough for us, as β-equivalent λ-terms have the same Böhm tree. ◀

It was shown [31, Theorem 3] that, for a closed λ-term K of sort o, the set $\mathcal{D}=\{A\subseteq\Sigma_{\mathcal{G}}\mid\mathsf{Diag}_A(\mathcal{L}(BT(K)))\}$ can be computed out of the value $[\![K]\!]$. We can thus easily convert the scheme $\mathcal{G}'$ from Lemma 7 to a scheme $\mathcal{G}_{diag}$ as needed in Theorem 6. Indeed, it is enough to replace, in every node constructor appearing in $\mathcal{G}'$, the pair (a,τ) by the pair $(a,\mathcal{D})$ for the set $\mathcal{D}$ computed out of the value τ.

5 Proof of the Main Theorem

In this section we prove our main theorem—Theorem 1. To this end, we have to recall two properties of recursion schemes: logical reflection, and closure under composition with finite tree transducers.

By MSO we mean the logic defined similarly to WMSO+U, but where there are no U quantifiers, and where existential quantifiers range over infinite sets. The MSO logic over infinite trees is equivalent to μ-calculus and to nondeterministic parity automata.

▶ **Fact 8** ([9, Theorem 2(ii)]). *For every scheme $\mathcal{G}$ generating a tree T and every MSO sentence φ one can construct a scheme $\mathcal{G}_\varphi$ that generates a tree of the same shape as T, and such that its every node v is labeled by a pair (a, b), where a is the label of v in T, and b is* tt *if φ is satisfied in $T{\restriction_v}$ and* ff *otherwise.*

A *(deterministic, top-down) finite tree transducer* is a tuple $\mathcal{T} = (Q, q_0, \Sigma_0, r_{\max}, \delta)$, where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\Sigma_0 \subseteq \Sigma$ is a finite alphabet, $r_{\max}$ is the maximal arity of considered trees, and δ is a transition function mapping $Q \times \Sigma_0 \times \{0, \ldots, r_{\max}\}$ to finite λ-terms. A triple (q, a, r) should be mapped by δ to a term that uses only node constructors and variables of the form $\mathsf{x}_{i,p}$, where $i \in \{1, \ldots, r\}$ and $p \in Q$ (applications and λ-binders are not allowed); at least one node constructor has to be used (the whole $\delta(q, a, r)$ cannot be equal to a variable).

For a $(\Sigma_0, r_{\max})$-tree T and a state $q \in Q$, we define $\mathcal{T}_q(T)$ by coinduction, as follows: if $T = a\langle T_1, \ldots, T_r\rangle$, then $\mathcal{T}_q(T)$ is the tree obtained from $\delta(q, a, r)$ by substituting $\mathcal{T}_p(T_i)$ for the variable $\mathsf{x}_{i,p}$, for all $i \in \{1, \ldots, r\}$ and $p \in Q$. In the root we start from the initial state, that is, we define $\mathcal{T}(T) = \mathcal{T}_{q_0}(T)$. We have the following fact.

▶ **Fact 9.** *For every scheme $\mathcal{G}$ generating a tree T, and for every finite tree transducer $\mathcal{T}$ one can construct a scheme $\mathcal{G}_\mathcal{T}$ that generates the tree $\mathcal{T}(T)$.*

This fact follows from the equivalence between schemes and collapsible pushdown systems [20], as it is straightforward to compose a collapsible pushdown system with $\mathcal{T}$ (where due to Fact 2 we can assume that $\Lambda(\mathcal{G})$ is fully convergent, i.e., that every node of T is explicitly generated by the collapsible pushdown system).

Having Facts 8 and 9, we now come to our main technical lemma.

▶ **Lemma 10.** *For every scheme $\mathcal{G}$ generating a tree T and every* U*-prefix automaton $\mathcal{A}$ one can construct a scheme $\mathcal{G}_\mathcal{A}$ that generates the tree $\mathcal{A}(T)$.*

It is easy to deduce Theorem 1 out of Lemma 10. Indeed, consider a WMSO+U sentence φ and a scheme $\mathcal{G}_0$ generating a tree T_0. By Lemma 3, φ is equivalent to a nested U-prefix automaton $\mathcal{A} = \mathcal{A}_1 \circ \cdots \circ \mathcal{A}_k$, together with an accepting set Σ_F. By consecutively applying Lemma 10 for $i = 1, \ldots, k$, we combine $\mathcal{G}_{i-1}$ with $\mathcal{A}_i$, obtaining a scheme $\mathcal{G}_i$ that generates the tree $T_i = \mathcal{A}_i(T_{i-1})$. The root of $T_k = \mathcal{A}(T_0)$ has label in Σ_F if and only if φ is satisfied in T_0. Surely this label can be read: having $\mathcal{G}_k$, we simply start generating the tree T_k, until its root is generated (by Fact 2, we can assume that $\Lambda(\mathcal{G}_k)$ is fully convergent).

We now come to the proof of Lemma 10. We are thus given a U-prefix automaton $\mathcal{A} = (Q, Q_\mathsf{imp}, \Delta)$, and a scheme $\mathcal{G}$ generating a tree T; our goal is to create a scheme $\mathcal{G}_\mathcal{A}$ that generates the tree $\mathcal{A}(T)$. As a first step, we create a finite tree transducer $\mathcal{T}$ that converts T into a tree containing all runs of $\mathcal{A}$ on all subtrees of T. Let us write $Q = \{p_1, \ldots, p_{|Q|}\}$. As $\mathcal{T}$ we take $(Q \cup \{q_0, \top\}, q_0, \Sigma_\mathcal{G}, r_{\max}(\mathcal{G}), \delta)$, where $q_0 \notin Q$ is a fresh state, and δ is defined as follows. For $q \in Q$, $a \in \Sigma_\mathcal{G}$, and $r \leq r_{\max}(\mathcal{G})$ we take

$$\delta(q, a, r) = \mathsf{nd}\langle q\langle \mathsf{x}_{1,q_{11}}, \ldots, \mathsf{x}_{r,q_{1r}}\rangle, \ldots, q\langle \mathsf{x}_{1,q_{k1}}, \ldots, \mathsf{x}_{r,q_{kr}}\rangle\rangle\,,$$

where $(q, a, q_{11}, \ldots, q_{1r}), \ldots, (q, a, q_{k1}, \ldots, q_{kr})$ are all elements of Δ being of length $r+2$ and having q and a on the first two coordinates. Moreover, for $a \in \Sigma_{\mathcal{G}}$ and $r \leq r_{\max}(\mathcal{G})$ we take

$$\delta(q_0, a, r) = a\langle \mathsf{x}_{1,q_0}, \ldots, \mathsf{x}_{r,q_0}, \delta(p_1, a, r), \ldots, \delta(p_{|Q|}, a, r)\rangle \qquad \text{and} \qquad \delta(\top, a, r) = \top\langle\rangle\,.$$

We see that $\mathcal{T}(T)$ contains all nodes of the original tree T. Additionally, below every node v coming from T we have $|Q|$ new children, such that subtrees starting in these children describe runs of $\mathcal{A}$ on $T{\restriction_v}$, starting in particular states. More precisely, when v has r children in T, for every $i \in \{1, \ldots, |Q|\}$ there is a bijection between trees U in $\mathcal{L}(\mathcal{T}(T){\restriction_{v(r+i)}})$ and runs ρ of $\mathcal{A}$ on $T{\restriction_v}$ such that $\rho(\varepsilon) = p_i$. The label of every node u in such a tree U contains the state assigned by ρ to u, where U contains exactly all nodes to which ρ assigns a state from Q, and all minimal nodes to which ρ assigns $\top$ (i.e., such that ρ does not assign $\top$ to their parents). Recall that by definition ρ can assign a state from Q only to a finite prefix of the tree $T{\restriction_v}$, which corresponds to the fact that $\mathcal{L}(\mathcal{T}(T){\restriction_{v(r+i)}})$ contains only finite trees.

Actually, we need to consider a transducer $\mathcal{T}'$ obtained from $\mathcal{T}$ by a slight modification: we replace the letter q appearing in $\delta(q, a, r)$ by 1 if $q \in Q_{\mathsf{imp}}$, and by 0 if $q \notin Q_{\mathsf{imp}}$. Then, for a node v of T having r children, and for $i \in \{1, \ldots, |Q|\}$, we have the following equivalence: $\mathsf{Diag}_{\{1\}}(\mathcal{T}'(T){\restriction_{v(r+i)}})$ holds if and only if for every $n \in \mathbb{N}$ there is a run ρ_n of $\mathcal{A}$ on $T{\restriction_v}$ that assigns p_i to the root of $T{\restriction_v}$, and such that for at least n nodes w it holds that $\rho_n(w) \in Q_{\mathsf{imp}}$.

We now apply Fact 9 to $\mathcal{G}$ and $\mathcal{T}'$; we obtain a scheme $\mathcal{G}_{\mathcal{T}'}$ that generates the tree $\mathcal{T}'(T)$. Then, we apply Theorem 6 (diagonal reflection) to $\mathcal{G}_{\mathcal{T}'}$, which gives us a scheme $\mathcal{G}'$. The tree T' generated by $\mathcal{G}'$ has the same shape as $\mathcal{T}'(T)$, but in the label of every node w there is additionally written a set $\mathcal{D}$ containing these sets $A \subseteq \Sigma$ for which $\mathsf{Diag}_A(\mathcal{L}(T{\restriction_w}))$ holds. Next, using Fact 8 (logical reflection) $2|Q|$ times, we annotate every node v of T', having r' children, by logical values of the following properties, for $i = 1, \ldots, |Q|$:

- whether $r' \geq |Q|$ and $\mathcal{L}(T'{\restriction_{v(r'-|Q|+i)}})$ is nonempty, and
- whether $r' \geq |Q|$ and the label $(a, \mathcal{D})$ of node $v(r' - |Q| + i)$ in T' satisfies $\{1\} \in \mathcal{D}$.

Clearly both these properties can be expressed in MSO. For nodes v coming from T, the first property holds when there is a run of $\mathcal{A}$ on $T{\restriction_v}$ that assigns p_i to the root of $T{\restriction_v}$, and the second property holds when for every $n \in \mathbb{N}$ there is a run ρ_n of $\mathcal{A}$ on $T{\restriction_v}$ that assigns p_i to the root of $T{\restriction_v}$, and such that for at least n nodes w it holds that $\rho_n(w) \in Q_{\mathsf{imp}}$. Let $\mathcal{G}''$ be the scheme generating the tree T'' containing these annotations.

Finally, we create $\mathcal{G}_{\mathcal{A}}$ by slightly modifying $\mathcal{G}''$: we replace every node constructor $(a, \mathcal{D}, \sigma_1, \tau_1, \ldots, \sigma_{|Q|}, \tau_{|Q|})\langle P_1, \ldots, P_{r+|Q|}\rangle$ with $f\langle P_1, \ldots, P_r\rangle$, where $f\colon Q \to \{0, 1, 2\}$ is such that $f(p_i) = 2$ if $\tau_i = \mathsf{tt}$, and $f(p_i) = 1$ if $\sigma_i = \mathsf{tt}$ but $\tau_i = \mathsf{ff}$, and $f(p_i) = 2$ otherwise, for all $i \in \{1, \ldots, |Q|\}$ (we do not do anything with node constructors of arity smaller than $|Q|$). In effect only the nodes coming from T remain, and they are appropriately relabeled.

6 Extensions

In this section we give a few possible extensions of our main theorem, saying that we can evaluate WMSO+U sentences on trees generated by recursion schemes. First, we notice that our solution actually proves a stronger result: logical reflection for WMSO+U.

▶ **Theorem 11.** *For every scheme $\mathcal{G}$ generating a tree T and every* WMSO+U *sentence φ one can construct a scheme $\mathcal{G}_\varphi$ that generates a tree of the same shape as T, and such that its every node v is labeled by a pair (a, b), where a is the label of v in T, and b is* tt *if φ is satisfied in $T{\restriction_v}$ and* ff *otherwise.*

Proof. In the proof of Theorem 1 we have constructed a nested U-prefix automaton $\mathcal{A}$ equivalent to φ, and then a scheme $\mathcal{G}_{\mathcal{A}}$ that generates the tree $\mathcal{A}(T)$. In every node v of $\mathcal{A}(T)$ it is written whether $T{\restriction_v}$ satisfies φ. Moreover, by appropriately altering $\mathcal{A}$, we can assume that labels of $\mathcal{A}(T)$ contain also original labels coming from T. Thus in order to obtain $\mathcal{G}_\varphi$ it is enough to appropriately relabel node constructors appearing in $\mathcal{G}_{\mathcal{A}}$. ◀

In Theorem 11, the formula φ talks only about the subtree starting in v. One can obtain a stronger version of logical reflection, where φ is allowed to talk about v in the context of the whole tree. This version can be obtained as a simple corollary of Theorem 11 by using the same methods as in Broadbent, Carayol, Ong, and Serre [9, Proof of Corollary 2].

▶ **Corollary 12.** *For every scheme $\mathcal{G}$ generating a tree T and every* WMSO+U *formula $\varphi(X)$ with one free variable X, one can construct a scheme $\mathcal{G}_\varphi$ that generates a tree of the same shape as T, and such that its every node v is labeled by a pair (a, b), where a is the label of v in T, and b is* tt *if φ is satisfied in T with X valuated to $\{v\}$, and* ff *otherwise.*

For MSO it is possible to prove another property, called effective selection [12]. This time we are given an MSO sentence φ of the form $\exists X.\psi$. Assuming that φ is satisfied in the tree T generated by a scheme $\mathcal{G}$, one wants to compute an example set X^T of nodes of T, such that ψ is true in T with the variable X valuated to this set X^T. In particular, it is possible to create a scheme $\mathcal{G}_\varphi$ which generates a tree of the same shape as T, in which nodes belonging to some such example set X^T are marked. In WMSO+U we can only quantify over finite sets, so the analogous property for $\varphi = \exists_{\mathsf{fin}} X.\psi$ can be trivially obtained (and hence it is not so interesting). Indeed, there are only countably many finite sets X^T, so we may try one after another, until we find some set for which ψ is satisfied; it is easy to hardcode a given set X^T in the formula (or in the scheme).

We notice that WMSO+U is incomparable to MSO, with respect to the expressive power. As model-checking of MSO sentences is also decidable on trees generated by schemes, we can consider a hybrid logic, covering both MSO and WMSO+U. To obtain such a logic, we introduce to WMSO+U quantifiers $\exists X$ ranging over infinite sets X, but with the requirement that if $\mathsf{U}Y.\psi$ is a subformula of $\exists X.\varphi$ then X is not a free variable of $\mathsf{U}Y.\psi$. In nested automata equivalent to sentences of this logic, beside of U-prefix automata (responsible for U quantifiers) we also have nondeterministic parity automata (responsible for subformulae using $\exists$ quantifiers). As we have the reflection property for both kinds of automata, our results generalize to this logic.

Our algorithm has nonelementary complexity. This is unavoidable, as already model-checking of WMSO sentences on the infinite word over an unary alphabet is nonelementary. It would be interesting to find some other formalism for expressing unboundedness properties, maybe using some model of automata, for which the model-checking problem has better complexity. We leave this issue for future work.

Finally, we remark that in our solution we do not use the full power of the diagonal problem, we only use the single-letter case. On the other hand, it seems that WMSO+U (and full MSO as well) is not capable to express the diagonal problem, only its single-letter case. Thus another direction for a future work is to extend WMSO+U to a logic that can actually express the diagonal problem. As a possible candidate we see the qcMSO logic introduced in Kaiser, Lang, Leßenich, and Löding [21], in which the diagonal problem is expressible.

References

1 Alfred V. Aho. Indexed grammars - an extension of context-free grammars. *J. ACM*, 15(4):647–671, 1968. doi:10.1145/321479.321488.

2 Achim Blumensath, Thomas Colcombet, and Christof Löding. Logical theories and compatible operations. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas].*, volume 2 of *Texts in Logic and Games*, pages 73–106. Amsterdam University Press, 2008.

3 Mikołaj Bojańczyk. A bounding quantifier. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic, 18th International Workshop, CSL 2004, 13th Annual Conference of the EACSL, Karpacz, Poland, September 20-24, 2004, Proceedings*, volume 3210 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2004. doi:10.1007/978-3-540-30124-0_7.

4 Mikołaj Bojańczyk. Weak MSO with the unbounding quantifier. *Theory Comput. Syst.*, 48(3):554–576, 2011. doi:10.1007/s00224-010-9279-2.

5 Mikołaj Bojańczyk. Weak MSO+U with path quantifiers over infinite trees. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 38–49. Springer, 2014. doi:10.1007/978-3-662-43951-7_4.

6 Mikołaj Bojańczyk, Pawel Parys, and Szymon Toruńczyk. The MSO+U theory of (n, <) is undecidable. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 21:1–21:8. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.21.

7 Mikołaj Bojańczyk and Szymon Toruńczyk. Weak MSO+U over infinite trees. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 648–660. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.648.

8 Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996. doi:10.1142/S0129054196000191.

9 Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. Recursion schemes and logical reflection. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 120–129. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.40.

10 Christopher H. Broadbent and Naoki Kobayashi. Saturation-based model checking of higher-order recursion schemes. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPIcs*, pages 129–148. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.CSL.2013.129.

11 Christopher H. Broadbent and C.-H. Luke Ong. On global model checking trees generated by higher-order recursion schemes. In Luca de Alfaro, editor, *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5504 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2009. doi:10.1007/978-3-642-00596-1_9.

12 Arnaud Carayol and Olivier Serre. Collapsible pushdown automata and labeled recursion schemes: Equivalence, safety and effective selection. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 165–174. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.73.

13 Lorenzo Clemente, Pawel Parys, Sylvain Salvati, and Igor Walukiewicz. Ordered tree-pushdown systems. In Prahladh Harsha and G. Ramalingam, editors, *35th IARCS An-*

nual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India, volume 45 of *LIPIcs*, pages 163–177. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.163.

14 Lorenzo Clemente, Pawel Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 96–105. ACM, 2016. doi:10.1145/2933575.2934527.

15 Wojciech Czerwinski, Wim Martens, Lorijn van Rooijen, and Marc Zeitoun. A note on decidable separability by piecewise testable languages. In Adrian Kosowski and Igor Walukiewicz, editors, *Fundamentals of Computation Theory - 20th International Symposium, FCT 2015, Gdańsk, Poland, August 17-19, 2015, Proceedings*, volume 9210 of *Lecture Notes in Computer Science*, pages 173–185. Springer, 2015. doi:10.1007/978-3-319-22177-9_14.

16 Werner Damm. The IO- and oi-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982. doi:10.1016/0304-3975(82)90009-3.

17 Solomon Feferman and Robert Lawson Vaught. The first order properties of products of algebraic systems. *Fundamenta Mathematicae*, 47(1):57–103, 1959. URL: http://eudml.org/doc/213526.

18 Tobias Ganzow and Lukasz Kaiser. New algorithm for weak monadic second-order logic on inductive structures. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 366–380. Springer, 2010. doi:10.1007/978-3-642-15205-4_29.

19 Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 151–163. ACM, 2016. doi:10.1145/2837614.2837627.

20 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 452–461. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.34.

21 Lukasz Kaiser, Martin Lang, Simon Leßenich, and Christof Löding. A unified approach to boundedness properties in MSO. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 441–456. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.CSL.2015.441.

22 Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002. doi:10.1007/3-540-45931-6_15.

23 Naoki Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In Martin Hofmann, editor, *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume

6604 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2011. doi:10.1007/978-3-642-19805-2_18.

24 Naoki Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20:1–20:62, 2013. doi:10.1145/2487241.2487246.

25 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 179–188. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.29.

26 Hans Läuchli. A decision procedure for the weak second order theory of linear order. *Studies in Logic and the Foundations of Mathematics*, 50:189–197, 1968. doi:10.1016/S0049-237X(08)70525-1.

27 Robin P. Neatherway and C.-H. Luke Ong. Travmc2: higher-order model checking for alternating parity tree automata. In Neha Rungta and Oksana Tkachuk, editors, *2014 International Symposium on Model Checking of Software, SPIN 2014, Proceedings, San Jose, CA, USA, July 21-23, 2014*, pages 129–132. ACM, 2014. doi:10.1145/2632362.2632381.

28 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 81–90. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.38.

29 Pawel Parys. Intersection types and counting. In Naoki Kobayashi, editor, *Proceedings Eighth Workshop on Intersection Types and Related Systems, ITRS 2016, Porto, Portugal, 26th June 2016.*, volume 242 of *EPTCS*, pages 48–63, 2016. doi:10.4204/EPTCS.242.6.

30 Pawel Parys and Szymon Toruńczyk. Models of lambda-calculus and the weak MSO logic. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France*, volume 62 of *LIPIcs*, pages 11:1–11:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CSL.2016.11.

31 Paweł Parys. Complexity of the diagonal problem for recursion schemes. Submitted to FSTTCS, 2017.

32 Steven J. Ramsay, Robin P. Neatherway, and C.-H. Luke Ong. A type-directed abstraction refinement approach to higher-order model checking. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 61–72. ACM, 2014. doi:10.1145/2535838.2535873.

33 Sylvain Salvati and Igor Walukiewicz. Using models to model-check recursive schemes. In Masahito Hasegawa, editor, *Typed Lambda Calculi and Applications, 11th International Conference, TLCA 2013, Eindhoven, The Netherlands, June 26-28, 2013. Proceedings*, volume 7941 of *Lecture Notes in Computer Science*, pages 189–204. Springer, 2013. doi:10.1007/978-3-642-38946-7_15.

34 Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. *Inf. Comput.*, 239:340–355, 2014. doi:10.1016/j.ic.2014.07.012.

35 Sylvain Salvati and Igor Walukiewicz. A model for behavioural properties of higher-order programs. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 229–243. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.CSL.2015.229.

36 Sylvain Salvati and Igor Walukiewicz. Simply typed fixpoint calculus and collapsible pushdown automata. *Mathematical Structures in Computer Science*, 26(7):1304–1350, 2016. doi:10.1017/S0960129514000590.

37 Saharon Shelah. The monadic theory of order. *Annals of Mathematics*, 102(3):379–419, 1975. doi:10.2307/1971037.

38 Georg Zetzsche. An approach to computing downward closures. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 440–451. Springer, 2015. doi:10.1007/978-3-662-47666-6_35.

Sums of Palindromes: an Approach via Automata

Aayush Rajasekaran
School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada
arajasekaran@uwaterloo.ca

Jeffrey Shallit
School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada
shallit@uwaterloo.ca

Tim Smith
School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada
timsmith@uwaterloo.ca

Abstract

Recently, Cilleruelo, Luca, and Baxter proved, for all bases $b \geq 5$, that every natural number is the sum of at most 3 natural numbers whose base-b representation is a palindrome. However, the cases $b = 2, 3, 4$ were left unresolved. We prove, using a decision procedure based on automata, that every natural number is the sum of at most 4 natural numbers whose base-2 representation is a palindrome. Here the constant 4 is optimal. We obtain similar results for bases 3 and 4, thus completely resolving the problem.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory, Mathematics of computing → Combinatorics on words, Theory of computation → Automated reasoning

Keywords and phrases Finite automaton, Nested-word Automaton, Decision Procedure, Palindrome, Additive Number Theory

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.54

Acknowledgements We thank Dirk Nowotka, Parthasarathy Madhusudan, and Jean-Paul Allouche for helpful discussions. We thank the creators of the ULTIMATE automaton library for their assistance. Finally, we thank the referees of STACS 2018 for their helpful comments and corrections.

1 Introduction

In this paper we develop a new method, based on automata theory, for solving problems in additive number theory. As an example of the power of our method, we are able to prove the new result that *every natural number is the sum of at most* 4 *numbers whose base*-2 *representation is a palindrome.*

Additive number theory is the study of the additive properties of integers; it has a very long and celebrated history. For example, Lagrange proved (1770) that every natural number is the sum of four squares (see, e.g., [12]). In additive number theory, a subset $S \subseteq \mathbb{N}$ is called an *additive basis of order* h if every element of $\mathbb{N}$ can be written as a sum of at most h members of S, not necessarily distinct.

Waring's problem asks for the smallest value $g(k)$ such that the k'th powers form a basis of order $g(k)$. Lagrange's theorem then shows that $g(2) = 4$. In a variation on Waring's problem, one can ask for the smallest value $G(k)$ such that every *sufficiently large* natural number is the sum of $G(k)$ k'th powers [26]. This kind of representation is called an *asymptotic additive basis* of order $G(k)$.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 54; pp. 54:1–54:12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


Quoting Nathanson [19, p. 7], *"The central problem in additive number theory is to determine if a given set of integers is a basis of finite order."*

In this paper we show how to solve this central problem for certain sets of natural numbers, using automata theory and *almost no number theory at all*.

We are concerned with numbers with particular representations in base k. For example, numbers of the form $11\cdots1$ in base k are sometimes called *repunits* [28], and special effort has been devoted to factoring such numbers, with the Mersenne numbers 2^n-1 being the most famous examples. The *Nagell-Ljunggren problem* asks for a characterization of those repunits that are integer powers (see, e.g., [24]).

Another interesting class, and the one that principally concerns us in this article, consists of those numbers whose base-k representation forms a *palindrome*: a string that reads the same forwards and backwards, like the English word `redder`, the French word `ressasser`, and the German word `reliefpfeiler`. Palindromic numbers have been studied for some time in number theory; see, for example, [7, 6], just to name two recent references.

Recently Banks initiated the study of the additive properties of palindromes, proving that every natural number is the sum of at most 49 numbers whose decimal representation is a palindrome [5]. Banks' result was then improved by Cilleruelo, Luca, and Baxter [8, 9], who proved that for all bases $b \geq 5$, every natural number is the sum of at most 3 numbers whose base-b representation is a palindrome. The proofs of Banks and Cilleruelo, Luca, and Baxter are both rather lengthy and case-based. Up to now, there have been no results proved for bases $b = 2, 3, 4$.

The long case-based solutions to the problem of representation by sums of palindromes suggests that perhaps a more automated approach might be useful. We turn to formal languages and automata theory as a suitable framework for expressing the palindrome representation problem. Since we want to make assertions about the representations of *all* natural numbers, this requires finding (a) a machine model or logical theory in which universality is decidable and (b) a variant of the additive problem of palindromes suitable for this machine model or logical theory. The first model we use is the *nested-word automaton*, a variant of the more familiar pushdown automaton. This is used to handle the case for base $b = 2$. The second model we use is the ordinary finite automaton, which we use to resolve the cases $b = 3, 4$.

Our paper is organized as follows: In Section 2 we introduce some notation and terminology, and state more precisely the problem we want to solve. In Section 3 we recall the pushdown automaton model and give an example, and we motivate our use of nested-word automata. In Section 4 we restate our problem in the framework of nested-word automata, and the proof of a bound of 4 palindromes is given in Section 5. In Section 6 we make use of finite automata to prove a bound of 3 palindromes for bases 3 and 4. The novelty of our approach involves replacing the long case-based reasoning of previous proofs with an automaton-based approach using a decision procedure. In Section 7 we discuss possible objections to our approach. Finally, in Section 8 we discuss future work.

2 The sum-of-palindromes problem

We first introduce some notation and terminology.

The natural numbers are $\mathbb{N} = \{0, 1, 2, \ldots\}$. If n is a natural number, then by $(n)_k$ we mean the string (or word) representing n in base k, with no leading zeroes, starting with the most significant digit. Thus, for example, $(43)_2 = 101011$. The alphabet Σ_k is defined to be $\{0, 1, \ldots, k-1\}$; by Σ_k^* we mean the set of all finite strings over Σ_k. If $x \in \Sigma_\ell^*$ for some

ℓ, then by $[x]_k$ we mean the integer represented by the string x, considered as if it were a number in base k, with the most significant digit at the left. That is, if $x = a_1 a_2 \cdots a_n$, then $[x]_k = \sum_{1 \le i \le n} a_i k^{n-i}$. For example, $[135]_2 = 15$.

If x is a string, then x^i denotes the string $\overbrace{xx\cdots x}^{i}$, and x^R denotes the reverse of x. Thus, for example, $(\texttt{ma})^2 = \texttt{mama}$, and $(\texttt{drawer})^R = \texttt{reward}$. If $x = x^R$, then x is said to be a *palindrome*.

We are interested in integers whose base-k representations are palindromes. In this article, we routinely abuse terminology by calling such an integer a *base-k palindrome*. In the case where $k = 2$, we also call such an integer a *binary palindrome*. The first few binary palindromes are

$$0, 1, 3, 5, 7, 9, 15, 17, 21, 27, 31, 33, 45, 51, 63, \ldots;$$

these form sequence A006995 in the *On-Line Encyclopedia of Integer Sequences* (OEIS).

If $k^{n-1} \le r < k^n$ for $n \ge 1$, we say that r is an *n-bit integer* in base k. If k is unspecified, we assume that $k = 2$. Note that the first bit of an n-bit integer is always nonzero. The *length* of an integer r satisfying $k^{n-1} \le r < k^n$ is defined to be n; alternatively, the length of r is $1 + \lfloor \log_k r \rfloor$.

Our goal is to find a constant c such that every natural number is the sum of at most c binary palindromes. To the best of our knowledge, no such bound has been proved up to now. In Sections 4 and 5 we describe how we used a decision procedure for nested-word automata to prove the following result:

▶ **Theorem 1.** *For all $n \ge 8$, every n-bit odd integer is either a binary palindrome itself, or the sum of three binary palindromes*
(a) *of lengths n, $n-2$, and $n-3$; or*
(b) *of lengths $n-1$, $n-2$, and $n-3$.*

As a corollary, we get our main result:

▶ **Corollary 2.** *Every natural number N is the sum of at most 4 binary palindromes.*

Proof. It is a routine computation to verify the result for $N < 128$.

Now suppose $N \ge 128$. Let N be an n-bit integer; then $n \ge 8$. If N is odd, then Theorem 1 states that N is the sum of at most 3 binary palindromes. Otherwise, N is even.

If $N = 2^{n-1}$, then it is the sum of $2^{n-1} - 1$ and 1, both of which are palindromes.

Otherwise, $N - 1$ is also an n-bit odd integer. Use Theorem 1 to find a representation for $N - 1$ as the sum of at most 3 binary palindromes, and then add the palindrome 1 to get a representation for N. ◀

▶ Remark. We note that the bound 4 is optimal since, for example, the number 176 is not the sum of three or fewer binary palindromes.

Sequence A261678 in the OEIS lists those even numbers that are not the sum of two binary palindromes. Sequence A261680 gives the number of distinct representations as the sum of four binary palindromes.

3 Finding an appropriate computational model

To find a suitable model for proving Theorem 1, we turn to formal languages and automata. We seek some class of automata with the following property: for each k, there is an automaton which, given a natural number n as input, accepts the input if and only if n can be expressed

as the sum of k palindromes. Furthermore, we would like the problem of universality ("Does the automaton accept every possible input?") to be decidable in our chosen model. By constructing the appropriate automaton and checking whether it is universal, we could then determine whether every number n can be expressed as the sum of k palindromes.

Palindromes suggest considering the model of pushdown automaton (PDA), since it is well-known that this class of machines, equipped with a stack, can accept the palindrome language $\texttt{PAL} = \{x \in \Sigma^* \ : \ x = x^R\}$ over any fixed alphabet Σ. A tentative approach is as follows: create a PDA M that, on input n expressed in base 2, uses nondeterminism to "guess" the k summands and verify that (a) every summand is a palindrome, and (2) they sum to the input n. We would then use a decision procedure for universality to determine whether M accepts all of its inputs. However, two problems immediately arise.

The first problem is that universality is recursively unsolvable for nondeterministic PDAs (see, e.g.,[15, Thm. 8.11, p. 203]), so even if the automaton M existed, there would be no algorithm guaranteed to check universality.

The second problem involves checking that the guessed summands are palindromes. One can imagine guessing the summands in parallel, or in series. If we try to check them in parallel, this seems to correspond to the recognition of a language which is not a CFL (i.e., a context-free language, the class of languages recognized by nondeterministic PDAs). Specifically, we encounter the following obstacle:

▶ **Theorem 3.** *The set of strings L over the alphabet $\Sigma \times (\Sigma \cup \{\#\})$, where the first "track" is a palindrome and the second "track" is another, possibly shorter, palindrome, padded on the right with # signs, is not a CFL.*

Proof. Assume that it is. Consider L intersected with the regular language

$$[1,1]^+[1,0][1,1]^+[0,1][1,\#]^+,$$

and call the result L'. We use Ogden's lemma [20] to show L' is not a CFL.

Let n be the constant in Ogden's lemma, and choose z to be the string where the first track is $(1^{2n}01^{2n})$ and the second track is $(1^n01^n\#^{2n})$. Mark the compound symbols $[1,\#]$. Then every factorization $z = uvwxy$ with vx nonempty must have at least one $[1,\#]$ in v or x. If it is in v, then the only choice for x is also $[1,\#]$, so pumping gives a non-palindrome on the first track. If it is in x then v can be $[1,1]^i$ or contain $[1,0]$ or $[0,1]$. If the latter, pumping twice gives a string not in L' because there is more than one 0 on one of the two tracks. If the former, pumping twice gives a string with the second track not a palindrome. This contradiction shows that L', and hence L, is not a context-free language. ◀

So, using a pushdown automaton, we cannot check whether two arbitrary strings of wildly unequal lengths, presented in parallel, are both palindromes.

If the summands were presented serially, we could check whether each summand individually is a palindrome, using the stack, but doing so destroys our copy of the summand, and so we cannot add them all up and compare them to the input. In fact, we cannot add serial summands in any case, because we have

▶ **Theorem 4.** *The language*

$$L = \{(m)_2\#(n)_2\#(m+n)_2 \ : \ m,n \geq 0\}$$

is not a CFL.

Proof. Assume L is a CFL and intersect with the regular language $1^+01^+\#1^+\#1^+0$, obtaining L'. We claim that

$$L' = \{1^a01^b\#1^c\#1^d0 \ : \ b = c \text{ and } a + b = d\}.$$

This amounts to the claim, easily verified, that the only solutions to the equation $2^{a+b+1} - 2^b - 1 + 2^c - 1 = 2^{d+1} - 2$ are $b = c$ and $a + b = d$. Then, if n is the constant in Ogden's lemma, starting with the string $z = 1^n01^n\#1^n\#1^{2n}0$, an easy argument proves that L' is not a CFL, and hence neither is L. ◀

So, using a pushdown automaton, we cannot handle summands presented in series, either.

These issues lead us to restrict our attention to representations as sums of palindromes of the same (or similar) lengths. More precisely, we consider the following variant of the additive problem of palindromes: for a length l and number of summands k, given a natural number n as input, is n the sum of k palindromes all of length exactly l? Since the palindromes are all of the same length, a stack would allow us to guess and verify them in parallel. To tackle this problem, we need a model which is both (1) powerful enough to handle our new variant, and (2) restricted enough that universality is decidable. We find such a model in the class of *nested-word automata*, described in the next section.

4 Restating the problem in the language of nested-word automata

Nested-word automata (NWAs) were popularized by Alur and Madhusudan [1, 2], although essentially the same model was discussed previously by Mehlhorn [18], von Braunmühl and Verbeek [27], and Dymond [10]. They are closely related to a restricted variant of pushdown automata called visibly-pushdown automata (VPAs). Under linear encodings, NWAs recognize the same class of languages as VPAs, namely the visibly-pushdown languages [1, 2]. We only briefly describe their functionality here. For other theoretical aspects of nested-word and visibly-pushdown automata, see [16, 22, 11, 23, 21]. The definition of NWAs provided here modifies the standard definition by borrowing some aspects of VPAs. We use this definition because that is what is used by the ULTIMATE program analysis framework [14, 13], which is the software tool we use to prove our results.

The input alphabet of an NWA is partitioned into three sets: a *call alphabet*, an *internal alphabet*, and a *return alphabet*. An NWA has a stack, but has more restricted access to it than PDAs do. If the input symbol read is from the call alphabet, the NWA pushes its current state onto the stack, and then performs a transition, based only on the current state and input symbol read. If an input symbol is from the internal alphabet, the NWA cannot access the stack in any way. If the input symbol read is from the return alphabet, the NWA pops the state at the top of the stack, and then performs a transition based on three pieces of information: the current state, the popped state, and the input state read. An NWA accepts if the state it terminates in is an accepting state.

As an example, Figure 1 illustrates a nested-word automaton accepting the language $\{0^n12^n \ : \ n \geq 1\}$. Here the call alphabet is $\{0\}$, the internal alphabet is $\{1\}$, and the return alphabet is $\{2\}$.

The first 0 pushes q_0 onto the stack. Each of the $n - 1$ subsequent 0s push q_1 onto the stack. When the machine reads the 1, it goes to state q_2 without accessing the stack. So long as the machine reads 2s and q_1 is on the stack, we stay in the non-accepting state q_2. Reading a 2 with q_0 on the top of the stack takes us to the only accepting state, q_3.

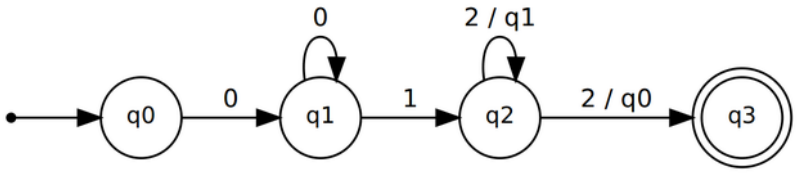


Figure 1 A nested-word automaton for the language $\{0^n 1 2^n \ : \ n \geq 1\}$.

Nondeterministic NWAs are a good machine model for our problem, because nondeterminism allows "guessing" the palindromes that might sum to the input, and the stack allows us to "verify" that they are indeed palindromes. Deterministic NWAs are as expressive as nondeterministic NWAs, and the class of languages they accept is closed under the operations of union, complement and intersection. Finally, testing emptiness, universality, and language inclusion are all decidable problems for NWAs [1, 2].

For a nondeterministic NWA of n states, the corresponding determinized machine has at most $2^{\Theta(n^2)}$ states, and there are examples for which this bound is attained. This very rapid explosion in state complexity potentially could make decision problems, such as language inclusion, infeasible in practice. Fortunately, we did not run into determinized machines with more than 40000 states in proving our results. Most of the algorithms invoked to prove our results run in under a minute.

We now discuss the general construction of the NWAs that check whether inputs are sums of binary palindromes. We partition the input alphabet into the call alphabet $\{a, b\}$, the internal alphabet $\{c, d\}$, and the return alphabet $\{e, f\}$. The symbols a, c, and e correspond to 0, while b, d, and f correspond to 1. The input string is fed to the machine starting with the *least significant digit*. We provide the NWA with input strings whose first half is entirely made of call symbols, and second half is entirely made of return symbols. Internal symbols are used to create a divider between the halves (for the case of odd-length inputs).

The idea behind the NWA is to nondeterministically guess all possible summands when reading the first half of the input string. The guessed summands are characterized by the states pushed onto the stack. The machine then checks if the guessed summands can produce the input bits in the second half of the string. The machine keeps track of any carries in the current state.

Following the above construction, we implemented our NWAs in the ULTIMATE program analysis framework. As an experimental spot check on the correctness of our implementations, we also built an NWA-simulator, and ran simulations of the machines on various types of inputs, which we then checked against experimental results.

For instance, we built the machine that accepts representations of integers that can be expressed as the sum of 2 binary palindromes. We then simulated this machine on every integer from 513 to 1024, and checked that it only accepts those integers that we experimentally confirmed as being the sums of 2 binary palindromes. This did not prove our results, but served as a sanity check.

The general procedure to prove our results is to build an NWA `PalSum` accepting only those inputs that it verifies as being appropriate sums of palindromes, as well as an NWA `SyntaxChecker` accepting all valid representations. We then run the decision algorithms

for language inclusion, language emptiness, etc. on `PalSum` and `SyntaxChecker` as needed. To do this, we used the Automata Library toolchain of the ULTIMATE program analysis framework.

We have provided links to the proof scripts used to establish all of our results. To run these proof scripts, simply copy the contents of the script into `https://monteverdi.informatik.uni-freiburg.de/tomcat/Website/?ui=int&tool=automata_library` and click "execute". Some of the larger proof scripts might time out on the web client, but can be run to completion by downloading the ULTIMATE Automata Library toolchain and executing them on a local machine.

5 Proving Theorem 1

In this section, we discuss construction of the appropriate nested-word automaton in more detail.

Proof of Theorem 1. We build three separate automata. The first, `palChecker`, has 9 states, and simply checks whether the input number is a binary palindrome. The second, `palChecker2`, has 771 states, and checks whether an input number of length n can be expressed as the sum of three binary palindromes of lengths n, $n-2$, and $n-3$. The third machine, `palChecker3`, has 1539 states, and checks whether an input number of length n can be expressed as the sum of three binary palindromes of lengths $n-1$, $n-2$, and $n-3$. We then determinize these three machines, and take their union, to get a single deterministic NWA, `FinalAut`, with 36194 states. We then run the command `FinalAut = shrinkNwa(FinalAut);` to reduce this to a deterministic NWA of only 106 states.

The language of valid inputs to our automata is given by

$$L = \{\{a,b\}^n\{c,d\}^m\{e,f\}^n \ : \ 0 \le m \le 1,\ n \ge 4\}.$$

We only detail the mechanism of `palChecker3` here. Let p, q and r be the binary palindromes representing the guessed $(n-1)$-length summand, $(n-2)$-length summand and $(n-3)$-length summand respectively. The states of `palChecker3` include 1536 t-states that are 10-tuples. We label these states $(g, x, y, z, k, l_1, l_2, m_1, m_2, m_3)$, where $0 \le g \le 2$, while all other coordinates are either 0 or 1. The g-coordinate indicates the current carry, and can be as large as 2. The x, y and z coordinates indicate whether we are going to guess 0 or 1 for the next guesses of p, q and r respectively. The remaining coordinates serve as "memory" to help us manage the differences in lengths of the guessed summands. The k-coordinate records the most recent guess for p. We have l_1 and l_2 record the two most recent q guesses, with l_1 being the most recent one, and we have m_1, m_2 and m_3 record the three most recent r-guesses, with m_1 being the most recent one, then m_2, and m_3 being the guess we made three steps ago. We also have three s-states labeled s_0, s_1 and s_2, representing carries of 0, 1 and 2 respectively. These states process the second half of the input string.

The initial state of the machine is $(0, 1, 1, 1, 0, 0, 0, 0, 0, 0)$ since we start with no carry, must guess 1 for our first guess of a valid binary palindrome, and all "previous" guesses are 0. A t-state has an outgoing transition on either a or b, but not both. If $g + x + y + z$ produces an output bit of 0, it takes a transition on a, else it takes a transition on b. The destination states are all six states of the form $(g', x', y', z', x, y, l_1, z, m_1, m_2)$, where g' is the carry resulting from $g + x + y + z$, and x', y', z' can be either 0 or 1. Note that we "update" the remembered states by forgetting k, l_2 and m_3, and saving x, y and z.

The s-states only have transitions on the return symbols e and f. When we read these symbols, we pop a t-state off the stack. If state s_i pops the state $(g, x, y, z, k, l_1, l_2, m_1, m_2, m_3)$

off the stack, its transition depends on the addition of $i + k + l_2 + m_3$. If this addition produces a carry of j, then s_i can take a transition to s_j on e if the output bit produced is 0, and on f otherwise. By reading the k, l_2 and m_3 values, we correctly realign p, q and r, correcting for their different lengths. This also ensures that we supply 0s for the last three guesses of r, the last two guesses of q and the last guess of p. The only accepting state is s_0.

It remains to describe how we transition from t-states to s-states. This transition happens when we are halfway done reading the input. If the length of the input is odd, we read a c or a d at the halfway point. We only allow certain t-states to have outgoing transitions on c and d. Specifically, we require the state's coordinates to satisfy $k = x$, $l_2 = y$, $m_1 = m_2$ and $m_3 = z$. These conditions are required for p, q and r to be palindromes. We transition to state s_i if the carry produced by adding $g + x + y + z$ is i, and we label the transition c if the output bit is 0, and d otherwise.

If the length of the input is even, then our t-states take a return transition on e or f. Once again, we restrict the t-states that can take return transitions. We require the state's coordinates to satisfy $l_1 = l_2$ and $m_1 = m_3$ to ensure our guessed summands are palindromes. Let the current state be $(g, x, y, z, k, l_1, l_1, m_1, m_2, m_1)$, and the state at the top of the stack be $(g', x', y', z', k', l'_1, l'_2, m'_1, m'_2, m'_3)$. We can take a transition to s_i if the sum $g+k'+l'_2+m'_3$ produces a carry of i, and we label the transition e if the output bit is 0, and f otherwise.

The structure and behavior of `palChecker2` is very similar. One difference is that there is no need for a k-coordinate in the t-states since the longest summand guessed is of the same length as the input.

The complete script executing this proof is over 750000 lines long. Since these automata are very large, we wrote two C++ programs to generate them. Both the proof script, and the programs generating them can be found at `https://cs.uwaterloo.ca/~shallit/papers.html`. It is worth noting that a t-state labeled as $(g, x, y, z, k, l_1, l_2, m_1, m_2, m_3)$ in this report is labeled $q\_g\_xyz\_k\_l_1l_2\_m_1m_2m_3$ in the proof script. Also, ULTIMATE does not currently have a union operation for NWAs, so we work around this by using De Morgan's laws for complement and intersection. ◀

6 Bases 3 and 4

In this section we prove analogous results for bases 3 and 4. We show that every natural number is the sum of at most three base-3 palindromes, and at most three base-4 palindromes. Because the NWAs needed became too large for us to manipulate effectively, we use a modified approach using nondeterministic finite automata to prove these results. This approach was suggested to us by Dirk Nowotka and Parthasarathy Madhusudan, independently.

Our result for base 3 is as follows:

▶ **Theorem 5.** *For all $n \geq 9$, every integer whose base-3 representation is of length n is the sum of*

(a) *three base-3 palindromes of lengths n, $n-1$, and $n-2$; or*
(b) *three base-3 palindromes of lengths n, $n-2$, and $n-3$; or*
(c) *three base-3 palindromes of lengths $n-1$, $n-2$, and $n-3$; or*
(d) *two base-3 palindromes of lengths $n-1$ and $n-2$.*

Proof. We represent the input in a "folded" manner over the input alphabet $\Sigma_3 \cup (\Sigma_3 \times \Sigma_3)$, where $\Sigma_k = \{0, 1, \ldots, k-1\}$, giving the machine 2 letters at a time from opposite ends. This way we can directly guess our summands without having need of a stack at all. We align the input along the length-$n-2$ summand by providing the first 2 letters of the input separately.

If $(N)_3 = a_{2i+1}a_{2i}\cdots a_0$, we represent N as the word

$$a_{2i+1}a_{2i}[a_{2i-1}, a_0][a_{2i-2}, a_1]\cdots[a_i, a_{i-1}].$$

Odd-length inputs leave a trailing unfolded letter at the end of their input. If $(N)_3 = a_{2i+2}a_{2i+1}\cdots a_0$, we represent N as the word

$$a_{2i+2}a_{2i+1}[a_{2i}, a_0][a_{2i-1}, a_1]\cdots[a_{i+1}, a_{i-1}]a_i.$$

We need to simultaneously carry out addition on both ends. In order to do this we need to keep track of two carries. On the lower end, we track the "incoming" carry at the start of an addition, as we did in the proofs using NWAs. On the higher end, however, we track the expected "outgoing" carry.

To illustrate how our machines work, we consider an NFA accepting length-n inputs that are the sum of 4 base-3 palindromes, one each of lengths n, $n-1$, $n-2$ and $n-3$. Although this is not a case in our theorem, each of the four cases in our theorem can be obtained from this machine by striking out one or more of the guessed summands.

Recall that we aligned our input along the length-$(n-2)$ summand by providing the 2 most significant letters in an unfolded manner. This means that our guesses for the length-n summand will be "off-by-two": when we make a guess at the higher end of the length-n palindromic summand, its appearance at the lower end is 2 steps away. We hence need to record the last 2 guesses at the higher end of the length-n summand in our state. Similarly, we need to record the most recent higher guess of the length-$(n-1)$ summand, since it is off by one. The length-$(n-2)$ summand is perfectly aligned, and hence nothing needs to be recorded. The length-$(n-3)$ summand has the opposite problem of the length-$(n-1)$ input. Its lower guess only appears at the higher end one step later, and so we save the most recent guess at the lower end.

Thus, in this machine, we keep track of 6 pieces of information:

- c_1, the carry we are expected to produce on the higher end,
- c_2, the carry we have entering the lower end,
- x_1 and x_2, the most recent higher guesses of the length-n summand,
- y, the most recent higher guess of the length-$(n-1)$ summand, and
- z, the most recent lower guess of the length-$(n-3)$ summand,

Consider a state $(c_1, c_2, x_1, x_2, y, z)$. Let $i, j, k, l \in [0, 2]$ be our next guesses for the four summands of lengths n, $n-1$, $n-2$ and $n-3$ respectively. Also, let α be our guess for the next incoming carry on the higher end. Let the result of adding $i+j+k+z+\alpha$ be a value $0 \le p_1 < 3$ and a carry of q_1. Let the result of adding $x+y+k+l+x_2$ be a value $0 \le p_2 < 3$ and a carry of q_2. We must have $q_1 = c_1$. If this condition is met, we add a transition from this state to $(\alpha, q_2, x_2, i, j, l)$, and label the transition $[p_1, p_2]$.

The initial state is $(0, 0, 0, 0, 0, 0)$. We expand the alphabet to include special variables for the first 3 symbols of the input string. This is to ensure that we always guess a 1 or a 2 for the first (and last) positions of our summands.

The acceptance conditions depend on whether $(N)_3$ is of even or odd length. If a state $(c_1, c_2, x_1, x_2, y, z)$ satisfies $c_1 = c_2$ and $x_1 = x_2$, we set it as an accepting states. A run can only terminate in one of these states if $(N)_3$ is of even length. We accept since we are confident that our guessed summands are palindromes (the condition $x_1 = x_2$ ensures our length-n summand is palindromic), and since the last outgoing carry on the lower end is the expected first incoming carry on the higher end (enforced by $c_1 = c_2$).

We also have a special symbol to indicate the trailing symbol of an input for which $(N)_3$ is of odd length. We add transitions from our states to a special accepting state, q_{acc}, if

we read this special symbol. Consider a state $(c_1, c_2, x_1, x_2, y, z)$, and let $0 \leq k < 3$ be our middle guess for the $n-2$ summand. Let the result of adding $x_1 + y + k + z + c_2$ be a value $0 \leq p < 3$ and a carry of q. If $q = c_1$, we add a transition on p from our state to q_{acc}.

We wrote a C++ program generating a single NFA with 4 parts, one for each case of the theorem. After minimizing, the machine has 378 states. We then built a second NFA that accepts folded representations of $(N)_3$ such that the unfolded length of $(N)_3$ is greater than 8. We then use ULTIMATE to assert that the language accepted by the second NFA is included in that accepted by the first. All these operations run in under a minute.

We tested this machine by experimentally calculating which values of $243 \leq N \leq 1000$ could be written as the sum of palindromes satisfying one of our 4 conditions. We then asserted that for all the folded representations of $243 \leq N \leq 1000$, our machine accepts these values which we experimentally calculated, and rejects all others. ◀

We also have the following result for base 4:

▶ **Theorem 6.** *For all $n \geq 7$, every integer whose base-4 representation is of length n is the sum of*

(a) *exactly one palindrome each of lengths $n-1$, $n-2$, and $n-3$; or*

(b) *exactly one palindrome each of lengths n, $n-2$, and $n-3$.*

Proof. The NFA we build is very similar to the machine described for the base-3 proof. Indeed, the generator used is the same as the one for the base-3 proof, except that its input base is 4, and the only machines it generates are for the two cases of this theorem. The minimized machine has 478 states. ◀

This, together with the results previously obtained by Cilleruelo, Luca, and Baxter, completes the additive theory of palindromes for all integer bases $b \geq 2$.

7 Objections to this kind of proof

A proof based on computer calculations, like the one we have presented here, is occasionally criticized because it cannot easily be verified by hand, and because it relies on software that has not been formally proved. These kinds of criticisms are not new; they date at least to the 1970's, in response to the celebrated proof of the four-color theorem by Appel and Haken [3, 4]. See, for example, Tymoczko [25].

We answer this criticism in several ways. First, it is not reasonable to expect that every result of interest to mathematicians will have short and simple proofs. There may well be, for example, easily-stated results for which the shortest proof possible in a given axiom system is longer than any human mathematician could verify in their lifetime, even if every waking hour were devoted to checking it. For these kinds of results, an automated checker may be our only hope. There are many results for which the only proof currently known is computational.

Second, while short proofs can easily be checked by hand, what guarantee is there that any very long case-based proof — whether constructed by humans or computers — can always be certified by human checkers with a high degree of confidence? There is always the potential that some case has been overlooked. Indeed, the original proof of the four-color theorem by Appel and Haken apparently overlooked some cases. Similarly, the original proof by Cilleruelo and Luca on sums of palindromes [8] had some minor flaws that became apparent once their method was implemented as a `python` program; these were later corrected in [9].

Third, confidence in the correctness of the results can be improved by providing code that others may check. Transparency is essential. To this end, we have provided our code for the nested-word automata, and the reader can easily run this code on the software we referenced.

8 Future work

We can use the same sorts of ideas to attack other problems in additive number theory. For example, we can obtain results about "generalized palindromes" (allowing an arbitrary number of leading zeroes), "anti-palindromes" (of the form $x\overline{x^R}$), "generalized anti-palindromes", and so forth.

In a recent paper [17], we use the same kinds of techniques to prove an analogue of Lagrange's theorem for binary "squares" (those numbers whose representation in base 2 consists of two consecutive identical blocks).

References

1 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM, 2004. doi:10.1145/1007352.1007390.

2 Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):16:1–16:43, 2009. doi:10.1145/1516512.1516518.

3 K. Appel and W. Haken. Every planar map is four colorable. I. Discharging. *Illinois J. Math.*, 21:429–490, 1977.

4 K. Appel, W. Haken, and J. Koch. Every planar map is four colorable. II. Reducibility. *Illinois J. Math.*, 21:491–567, 1977.

5 W. D. Banks. Every natural number is the sum of forty-nine palindromes. *INTEGERS — Electronic J. Combinat. Number Theory*, 16, 2016. #A3.

6 W. D. Banks and I. E. Shparlinski. Average value of the Euler function on binary palindromes. *Bull. Pol. Acad. Sci. Math.*, 54:95–101, 2006. doi:10.4064/ba54-2-1.

7 William D. Banks and Igor E. Shparlinski. Prime divisors of palindromes. *Periodica Mathematica Hungarica*, 51(1):1–10, 2005. doi:10.1007/s10998-005-0016-6.

8 J. Cilleruelo and F. Luca. Every positive integer is a sum of three palindromes. Preprint available at https://arxiv.org/abs/1602.06208v1, 2016.

9 J. Cilleruelo, F. Luca, and L. Baxter. Every positive integer is a sum of three palindromes. *Math. Comp.*, 2017. doi:10.1090/mcom/3221.

10 Patrick W. Dymond. Input-driven languages are in log n depth. *Inf. Process. Lett.*, 26(5):247–250, 1988. doi:10.1016/0020-0190(88)90148-2.

11 Yo-Sub Han and Kai Salomaa. Nondeterministic state complexity of nested word automata. *Theor. Comput. Sci.*, 410(30-32):2961–2971, 2009. doi:10.1016/j.tcs.2009.01.004.

12 G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, 5th edition, 1985.

13 Matthias Heizmann, Daniel Dietsch, Marius Greitschus, Jan Leike, Betim Musa, Claus Schätzle, and Andreas Podelski. Ultimate automizer with two-track proofs - (competition contribution). In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9636 of *Lecture Notes in Computer Science*, pages 950–953. Springer, 2016. doi:10.1007/978-3-662-49674-9_68.

14 Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Software model checking for people who love automata. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 36–52. Springer, 2013. doi:10.1007/978-3-642-39799-8_2.

15 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

16 Salvatore La Torre, Margherita Napoli, and Mimmo Parente. On the membership problem for visibly pushdown languages. In Susanne Graf and Wenhui Zhang, editors, *Automated Technology for Verification and Analysis, 4th International Symposium, ATVA 2006, Beijing, China, October 23-26, 2006.*, volume 4218 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2006. doi:10.1007/11901914_10.

17 P. Madhusudan, D. Nowotka, A. Rajasekaran, and J. Shallit. Lagrange's theorem for binary squares. Preprint available at https://arxiv.org/abs/1710.04247, 2017.

18 Kurt Mehlhorn. Pebbling moutain ranges and its application of dcfl-recognition. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherland, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 422–435. Springer, 1980. doi:10.1007/3-540-10003-2_89.

19 M. B. Nathanson. *Additive Number Theory: The Classical Bases*. Springer-Verlag, 1996.

20 William F. Ogden. A helpful result for proving inherent ambiguity. *Mathematical Systems Theory*, 2(3):191–194, 1968. doi:10.1007/BF01694004.

21 Alexander Okhotin and Kai Salomaa. State complexity of operations on input-driven pushdown automata. *J. Comput. Syst. Sci.*, 86:207–228, 2017. doi:10.1016/j.jcss.2017.02.001.

22 Xiaoxue Piao and Kai Salomaa. Operational state complexity of nested word automata. *Theor. Comput. Sci.*, 410(35):3290–3302, 2009. doi:10.1016/j.tcs.2009.05.002.

23 Kai Salomaa. Limitations of lower bound methods for deterministic nested word automata. *Inf. Comput.*, 209(3):580–589, 2011. doi:10.1016/j.ic.2010.11.021.

24 T. N. Shorey. On the equation $z^q = (x^n - 1)/(x - 1)$. *Indag. Math.*, 48:345–351, 1986. doi:10.1016/1385-7258(86)90020-X.

25 T. Tymoczko. The four-color problem and its philosophical significance. *J. Philosophy*, 76(2):57–83, 1979.

26 R. C. Vaughan and T. Wooley. Waring's problem: a survey. In M. A. Bennett, B. C. Berndt, N. Boston, H. G. Diamond, A. J. Hildebrand, and W. Philipp, editors, *Number Theory for the Millennium. III*, pages 301–340. A. K. Peters, 2002.

27 Burchard von Braunmühl and Rutger Verbeek. Input-driven languages are recognized in log n space. In Marek Karpinski, editor, *Fundamentals of Computation Theory, Proceedings of the 1983 International FCT-Conference, Borgholm, Sweden, August 21-27, 1983*, volume 158 of *Lecture Notes in Computer Science*, pages 40–51. Springer, 1983. doi:10.1007/3-540-12689-9_92.

28 S. Yates. The mystique of repunits. *Math. Mag.*, 51:22–28, 1978.

On the Containment Problem for Linear Sets

Hans U. Simon
Department of Mathematics, Ruhr-University Bochum, Germany
hans.simon@rub.de

Abstract

It is well known that the containment problem (as well as the equivalence problem) for semilinear sets is log-complete in Π_2^p (where hardness even holds in dimension 1). It had been shown quite recently that already the containment problem for multi-dimensional linear sets is log-complete in Π_2^p (where hardness even holds for a unary encoding of the numerical input parameters). In this paper, we show that already the containment problem for 1-dimensional linear sets (with binary encoding of the numerical input parameters) is log-hard (and therefore also log-complete) in Π_2^p. However, combining both restrictions (dimension 1 and unary encoding), the problem becomes solvable in polynomial time.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Polynomial Hierarchy, Completeness, Containment Problem, Linear Sets

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.55

Acknowledgements I want to thank the reviewers of this paper for their valuable suggestions. Many thanks go to Dmitry Chistikov and Christoph Haase who pointed my attention to [2], a paper that (without mentioning this explicitly) yields the log-hardness of the containment problem for linear sets of variable dimension.

1 Introduction

The containment problem for a family of sets consists in finding an answer to the following question: given two sets of the family, is the first one a subset of the second one?

It had been shown in a very early stage of complexity theory that the containment and the equivalence problem for semilinear sets are log-complete in Π_2^p (the second level of the polynomial hierarchy) [4]. This early investigation had been motivated by the fact that, first, the equivalence problem for contextfree languages is recursively undecidable and, second, the commutative images of contextfree languages happen to be semilinear sets according to Parikh's theorem [5]. Showing inequivalence of the commutative images of two given contextfree languages would therefore demonstrate their inequivalence.

Linear sets are the basic building blocks of semilinear sets. (The latter are finite unions of linear sets.) Moreover, 1-dimensional linear sets are the central object of research in the study of numerical semigroups [6]. It was shown quite recently that the containment problem for linear sets of variable dimension is log-complete in Π_2^p, where hardness even holds when numbers are encoded in unary [2]. In this paper, we extend the latter result as follows:

1. The containment problem for 1-dimensional linear sets (with a binary encoding of numbers) is log-hard (and therefore also log-complete) in Π_2^p.
2. On the other hand, the containment problem for 1-dimensional linear sets with a unary encoding of numbers becomes is solvable in polynomial time.

Moreover, in order to prove these results, we show the following:

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 55; pp. 55:1–55:12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


- The containment problem for so-called simple unary $(+, \cup)$-expressions[1] is log-hard in Π_2^p.
- The containment problem for linear sets is still log-hard in Π_2^p under a relatively strong promise. See Sections 2.5 and 3 for details.

These results might be of independent interest.

As for semilinear sets, the containment and the inequivalence problem have the same inherent complexity: both are log-complete in Π_2^p. We briefly note that the situation is different for linear sets. The equivalence problem for linear sets is easily shown to be computationally equivalent to the word problem for linear sets, and the latter is easily shown to be NP-complete. Hence, for linear sets, verifying containment is much harder than verifying equivalence.

This paper is structured as follows. In Section 2 we present the basic definitions and notations, and we mention some facts. Our main results are stated and proved in Section 3. One of these proofs is however postponed to the final Section 4 because it is a suitable modification of a similar proof of Stockmeyer (and is given for the sake of completeness). In the final Section 5, an open problem is mentioned.

2 Definitions, Notations and Facts

We assume familiarity with basic concepts from complexity theory (e.g., logspace reductions, log-hardness or log-completeness, polynomial hierarchy etc.). The complexity classes of the polynomial hierarchy will be denoted, as usual, by Σ_k^p and Π_k^p for $k = 0, 1, 2, \ldots$. We will mainly deal with the class Π_2^p on the second level of the hierarchy.

In Section 2.1, we briefly call into mind the definition of true quantified Boolean formulas which give rise to a hierarchy of problems with one log-complete problem at every level of the polynomial hierarchy. Section 2.2 contains the basic definitions that we need in connection with integer expressions. In Section 2.3, we briefly remind the reader to the definition of linear and semilinear sets. Some well known results on the inherent complexity of the containment problem for integer expressions resp. for semilinear sets are mentioned in Section 2.4. Section 2.5 briefly calls into mind the notion of promise problems.

2.1 Quantified Boolean Formulas

▶ **Definition 1** ([7]). Let $X_1, X_2, \ldots, X_k$ with $X_i = \{x_{i1}, x_{i2}, \ldots\}$ be disjoint collections of Boolean variables. Let $f(X_1, \ldots, X_k)$ denote any Boolean formula over (finitely many of) the variables from $X_1 \cup \ldots \cup X_k$. Let $Q_k = \exists$ if $k \geq 1$ is odd and $Q_k = \forall$ if $k \geq 1$ is even. The notation "$\exists X_i : \ldots$" means "there exists an assignment of the variables in X_i such that $\ldots$". The analogous remark applies to the notation "$\forall X_i : \ldots$". Given these notations, we define

$$\mathcal{B}_k = \{f(X_1, \ldots, X_k) : \ (\exists X_1, \forall X_2, \ldots, Q_k X_k : f(X_1, \ldots, X_k) = 1\} \ .$$

The set consisting of Boolean formulas $f(X_1, \ldots, X_k)$ outside of $\mathcal{B}_k$ is denoted as $\overline{\mathcal{B}_k}$. The subproblem of $\mathcal{B}_k$ (resp. of $\overline{\mathcal{B}_k}$) with f being a formula in conjunctive normal form is denoted as $\mathcal{B}_k^{CNF}$ (resp. as $\overline{\mathcal{B}_k}^{CNF}$). The corresponding subproblems with f being a formula in disjunctive normal form are denoted as $\mathcal{B}_k^{DNF}$ and $\overline{\mathcal{B}_k}^{DNF}$, respectively.

[1] a variant of a problem that has originally been analyzed by Stockmeyer [7]

▶ **Theorem 2** ([7]). *For any $k \geq 1$, $\mathcal{B}_k$ is log-complete in Σ_k^p. The same is true for $\mathcal{B}_k^{CNF}$ if k is odd and for $\mathcal{B}_k^{DNF}$ if k is even.*

▶ **Corollary 3.** *For any $k \geq 1$, $\overline{\mathcal{B}_k}$ is log-complete in Π_k^p. This even holds for the set $\overline{\mathcal{B}_k}^{CNF}$ if $k \geq 1$ is odd and for the set $\overline{B_k}^{DNF}$ if $k \geq 1$ is even.*

▶ **Example 4.** The set $\overline{\mathcal{B}_2}^{DNF}$, which coincides with the set of all Boolean DNF-formulas $f(X_1, X_2)$ satisfying

$$\forall X_1, \exists X_2 : f(X_1, X_2) = 0 \ .$$

is log-complete in Π_2^p.

2.2 Integer Expressions

▶ **Definition 5.** Let $m \geq 1$ be a positive integer. The set $\mathcal{E}_m$ of *m-dimensional unary integer expressions*, simply called *unary integer expressions* if m is clear from context, is the smallest set with the following properties:

1. $\{0,1\}^m \subseteq \mathcal{E}_m$. The tuples $(b_1, \ldots, b_m) \in \{0,1\}^m$ are called *atomic expressions*.
2. For any $E_1, E_2 \in \mathcal{E}_m$: $(E_1 \cup E_2), (E_1 + E_2) \in \mathcal{E}_m$.

Every expression $E \in \mathcal{E}_m$ represents a set $L(E) \subseteq \mathbb{N}_0^m$ that is defined in the obvious manner.

We briefly note that the classical definition of integer expressions in [7] is different from ours: there the expressions define subsets of $\mathbb{N}_0$, and an atomic expression is a binary representation of a single number in $\mathbb{N}_0$. In other words, the classical definition deals with 1-dimensional binary expressions whereas we deal with multi-dimensional unary expressions.

Since "$\cup$" is an associative operation, we may simply write $(E_1 \cup E_2 \cup E_3 \cup \ldots \cup E_s)$ instead of $(\ldots((E_1 \cup E_2) \cup E_3) \cup \ldots \cup E_s)$. The analogous remark applies to the operation "$+$".

▶ **Definition 6.** An expression $E \in \mathcal{E}_m$ is said to be a *$(+,\cup)$-expression* if it is a sum of unions of atomic expressions. A $(+,\cup)$-expression is called *simple* if every union in the sum is the union of precisely two (not necessarily different) atomic expressions.

▶ **Example 7.** The string

$$E = ((1,1,0) \cup (0,0,0)) + ((1,0,0) \cup (1,0,0)) + ((1,1,1) \cup (0,0,0))$$

is a simple unary $(+,\cup)$-expression. It represents the set

$$L(E) = \{(1,0,0), (2,1,0), (2,1,1), (3,2,1)\} \ .$$

2.3 Linear and Semilinear Sets

▶ **Definition 8.** The set $L(\mathbf{c}, P) \subseteq \mathbb{N}_0^m$ induced by $\mathbf{c} \in \mathbb{N}_0^m$ and a finite set $P = \{\mathbf{p_1}, \ldots, \mathbf{p_k}\} \subset \mathbb{N}_0^m$ is defined as

$$L(\mathbf{c}, P) = \mathbf{c} + \langle P \rangle \quad \text{where} \quad \langle P \rangle = \left\{ \sum_{i=1}^{k} a_i \cdot \mathbf{p_i} : a_i \in \mathbb{N}_0 \right\} \ .$$

The elements in P are called *periods* and $\mathbf{c}$ is called the *constant vector* of $L(\mathbf{c}, P)$. A subset L of $\mathbb{N}_0^m$ is called *linear* if $L = L(\mathbf{c}, P)$ for some $\mathbf{c} \in \mathbb{N}_0^m$ and some finite set $P \subset \mathbb{N}_0^m$. A *semilinear set* in $\mathbb{N}_0^m$ is a finite union of linear sets in $\mathbb{N}_0^m$.

2.4 Containment Problems

As mentioned already in the introduction, the *containment problem* for a family of sets consists in finding an answer to the following question: given two sets of the family, is the first one a subset of the second one? We will be mainly concerned with the containment problem for integer expressions and with the containment problem for linear and semilinear sets. We will assume that the dimension m of sets in $\mathbb{N}_0^m$ is part of the input unless we explicitly talk about an m-dimensional problem for some fixed constant m. The following is known:

1. The containment problem for 1-dimensional binary integer expressions is log-complete in Π_2^p [7].
2. The containment problem for semilinear sets is log-complete in Π_2^p [4]. The log-hardness in Π_2^p even holds either when numbers are encoded in unary or when the dimension is fixed to 1.
3. The containment problem for linear sets is log-complete in Π_2^p [2]. The log-hardness in Π_2^p even holds when numbers are encoded in unary.

The first two hardness results are shown by means of a logspace reduction from $\overline{\mathcal{B}_2}^{DNF}$ to the respective containment problem. A suitable modification of Stockmeyer's reduction from $\overline{\mathcal{B}_2}^{DNF}$ to the containment problem for 1-dimensional binary integer expressions leads to the following result:

▶ **Theorem 9.** *The containment problem for simple unary* $(+, \cup)$*-expressions is* log*-complete in* Π_2^p*.*

The proof of this theorem will be given in Section 4.

Notation for Vectors: The j-th component of a vector $\mathbf{x}$ is denoted as x_j or, occasionally, as $\mathbf{x}[j]$. The latter notation is used, for instance, if there is a sequence of vectors, say $\mathbf{x_1}, \ldots, \mathbf{x_n}$. The j-th component of $\mathbf{x_i}$ is then denoted as $\mathbf{x_i}[j]$ (as opposed to $x_{i,j}$ or $(x_i)_j$). Throughout the paper, we use $\mathbf{a^m}$ (with $a \in \mathbb{N}_0$) as a short notation for $(a, \ldots, a) \in \mathbb{N}_0^m$. For instance $\mathbf{1^m}$ denotes the all-ones vector in $\mathbb{N}_0^m$. The vector with value 1 in the i-th component and zeros in the remaining $m-1$ components is denoted as $\mathbf{e_i^m}$.

2.5 Promise Problems

A decision problem (without promise) is a problem with "yes"- and "no"-instances. A *promise problem* is a decision problem augmented by a promise that the input instances passed to an algorithm satisfy a certain condition. An algorithm needs to solve the promise problem only on the input instances that satisfy this condition. It may output anything on the remaining instances. Hence a promise problem has besides the "yes"- and the "no"-instances a third kind of instances: the ones that violate the promised condition. Decision problem can be viewed as promise problems with an empty promise. Reductions between promise problems should map "yes"-instances (resp. "no"-instances) of the first problem to "yes"-instances (resp. "no"-instances) of the second problem.

3 Main Results

The first result in this section will be concerned with the containment problem for linear sets when the latter is viewed as the following promise problem.

Instance: dimension m, finite sets $P, Q \subset \mathbb{N}_0^m$, vectors $\mathbf{c}, \mathbf{d} \in \mathbb{N}_0^m$ and $s \in \{1, \ldots, |P|\}$.

Question: $L(\mathbf{c}, P) \subseteq L(\mathbf{d}, Q)$?

Promise: Let $P = \{\mathbf{p_1}, \ldots, \mathbf{p_k}\}$ and let $K_s = \left\{a \in \{0,1\}^k \;\middle|\; \sum_{i=1}^k a_i = s\right\}$. With this notation, the following holds:

$$\forall a \in \mathbb{N}_0^k \setminus K_s : \sum_{i=1}^k a_i \cdot \mathbf{p_i} \in L(\mathbf{d}, Q) \ . \tag{1}$$

In other words: we make the promise that the inclusion $L(\mathbf{c}, P) \subseteq L(\mathbf{d}, Q)$ can possibly fail only on linear combinations of $\mathbf{p_1}, \ldots, \mathbf{p_k}$ with coefficient vectors taken from K_s.

In [2], it was shown that the containment problem for linear sets is log-hard in Π_2^p. We strengthen this result by showing that even the corresponding promise problem exhibits this kind of hardness. This slightly stronger result will later help us to prove the hardness of the containment problem for 1-dimensional linear sets.

▶ **Theorem 10.** *The containment problem for linear sets is* log*-hard in* Π_2^p *even under the promise (1) and even when numbers are encoded in unary.*

Proof. We will describe a logspace reduction from the containment problem for simple unary $(+, \cup)$-expressions to the containment problem for linear sets. An instance of the former problem is of the form

$$E = \sum_{i=1}^{s} (\mathbf{B_{i1}} \cup \mathbf{B_{i2}}) \quad \text{and} \quad E' = \sum_{i=1}^{s'} (\mathbf{B'_{i1}} \cup \mathbf{B'_{i2}}) \tag{2}$$

where $\mathbf{B_{i1}}, \mathbf{B_{i2}}, \mathbf{B'_{i1}}, \mathbf{B'_{i2}} \in \{0,1\}^m$. Note that we may set $s' = s$ because we could add sum-terms of the form $(\mathbf{0^m} \cup \mathbf{0^m})$ to the expression which has fewer terms. Our goal is to design $(2m + 2s)$-dimensional linear sets $\mathbf{c} + \langle P \rangle$ and $\langle P' \cup P'' \rangle$ such that

$$L(E) \subseteq L(E') \Leftrightarrow \mathbf{c} + \langle P \rangle \subseteq \langle P' \cup P'' \rangle \ . \tag{3}$$

Intuitively, we should think of vectors from $\mathbb{N}_0^{2m+2s}$ as being decomposed into four sections of dimension m, s, s, m, respectively. The first section is called the "base section"; the latter three are called "control sections". The constant vector $\mathbf{c}$ and the periods in $P = \{\mathbf{p_{ij}} : i \in [s], j \in [2]\}$ are chosen as follows:

$$\mathbf{c} = (\mathbf{0^m}, \mathbf{2^s}, \mathbf{1^s}, \mathbf{1^m}) \quad \text{and} \quad \mathbf{p_{ij}} = (\mathbf{B_{ij}}, \mathbf{e_i^s}, \mathbf{0^s}, \mathbf{0^m}) \ . \tag{4}$$

Note that the base section of the periods in P contains the atomic sub-expressions of E. The vectors in $\mathbb{N}_0^{2m+2s}$ having $(\mathbf{3^s}, \mathbf{1^s}, \mathbf{1^m})$ in their control sections are said to be "essential". It is evident that

$$L(E) \times \{3\}^s \times \{1\}^s \times \{1\}^m = (\mathbf{c} + \langle P \rangle) \cap (\mathbb{N}_0^m \times \{3\}^s \times \{1\}^s \times \{1\}^m) \ .$$

In other words: the set of base sections of the essential vectors in $\mathbf{c} + \langle P \rangle$ coincides with $L(E)$. The periods in $P' = \{\mathbf{p'_{ij}} : i \in [s], j \in [2]\}$ are similarly defined as the periods in P:

$$\mathbf{p'_{ij}} = \begin{cases} (\mathbf{B'_{ij}}, 3 \cdot \mathbf{e_i^s}, \mathbf{e_i^s}, \mathbf{0^m}) & \text{if } i \in [s-1] \\ (\mathbf{B'_{sj}}, 3 \cdot \mathbf{e_s^s}, \mathbf{e_i^s}, \mathbf{1^m}) & \text{if } i = s \end{cases} \ .$$

Clearly,

$$L(E') \times \{3\}^s \times \{1\}^s \times \{1\}^m = \langle P' \rangle \cap (\mathbb{N}_0^m \times \{3\}^s \times \{1\}^s \times \{1\}^m) \ .$$

Note that $L(E) \subseteq L(E')$ iff any essential vector in $\mathbf{c} + \langle P \rangle$ is contained in $\langle P' \rangle$. In order to get the desired equivalence (3), we will design P'' such that the following holds:

Claim 1: Any inessential vector from $\mathbf{c} + \langle P \rangle$ is contained in $\langle P'' \rangle$.

Claim 2: Any essential vector in $\mathbf{c} + \langle P \rangle$ is contained in $\langle P' \cup P'' \rangle$ only if it is already contained in $\langle P' \rangle$.

It is evident that (3) is valid if P'' can be defined in accordance with the two above claims. Let $n = 1 + \max\{x_i : \mathbf{x} \in L(E), i \in [m]\}$, i.e., $n-1$ is the largest number that occurs in a component of some vector in $L(E)$. We now set $P'' = P_1'' \cup P_2''$ where

$$\begin{aligned} P_1'' &= \{(\mathbf{0^m}, 2 \cdot \mathbf{e_i^s}, \mathbf{1^s}, \mathbf{0^m}), (\mathbf{0^m}, 2 \cdot \mathbf{e_i^s}, \mathbf{0^s}, \mathbf{0^m}), (\mathbf{0^m}, 3 \cdot \mathbf{e_i^s}, \mathbf{0^s}, \mathbf{0^m}) : i \in [s]\} \ , \\ P_2'' &= \{(r \cdot \mathbf{e_i^m}, \mathbf{0^s}, \mathbf{0^s}, \mathbf{e_i^m}), (n \cdot \mathbf{e_i^m}, \mathbf{0^s}, \mathbf{0^s}, \mathbf{0^m}) : i \in [m], r \in \{0, 1, \ldots, n-1\}\} \ . \end{aligned}$$

The proof of the theorem can now be accomplished by showing that the above two claims are valid for our definition of P'' (and by adding some easy observations).

Proof of Claim 1: Let $\mathbf{x} \in \mathbf{c} + \langle P \rangle$ be inessential. An inspection of (4) reveals that there must exist an index $i_0 \in [s]$ such that the i_0-th component of the first control section of $\mathbf{x}$ has a value that differs from 3. Since already the constant vector $\mathbf{c}$ makes a contribution of 2 in this control section, the possible values for x_{m+i_0} are $2, 4, 5, 6, \ldots$. In order to cast $\mathbf{x}$ as a member of $\langle P'' \rangle$, we first pick the vector $\mathbf{u} = (\mathbf{0^m}, 2 \cdot \mathbf{e_{i_0}^s}, \mathbf{1^s}, \mathbf{0^m})$. Note that $\mathbf{u} \leq \mathbf{x}$ and $\mathbf{u}$ already coincides with $\mathbf{x}$ in the second control section. Adding to $\mathbf{u}$ properly chosen multiples of vectors of the form $(\mathbf{0^m}, 2 \cdot \mathbf{e_i^s}, \mathbf{0^s}, \mathbf{0^m})$ or $(\mathbf{0^m}, 3 \cdot \mathbf{e_i^s}, \mathbf{0^s}, \mathbf{0^m})$, we obtain a vector $\mathbf{v} \leq \mathbf{x}$ that coincides with $\mathbf{x}$ also in the first control section. Consider now the entries of $\mathbf{v}$ and $\mathbf{x}$ in the base section. For any $i \in [m]$, consider the decomposition $x_i - v_i = q_i n + r_i$ with $q_i \geq 0$ and $0 \leq r_i \leq n-1$. Adding to v the vector

$$\sum_{i=1}^{m} (q_i \cdot (n \cdot \mathbf{e_i^m}, \mathbf{0^s}, \mathbf{0^s}, \mathbf{0^m}) + (r_i \cdot \mathbf{e_i^m}, \mathbf{0^s}, \mathbf{0^s}, \mathbf{e_i^m})) \ ,$$

we obtain a vector that coincides with $\mathbf{x}$ (since, by now, it also coincides with $\mathbf{x}$ in the base section and in the third control section).

Proof of Claim 2: Let $\mathbf{x} \in \mathbf{c} + \langle P \rangle$ be essential and suppose that $\mathbf{x} \in \langle P' \cup P'' \rangle$. A representation of $\mathbf{x}$ as a member of $\langle P' \cup P'' \rangle$ cannot make use of a vector of the form $(\mathbf{0^m}, 2 \cdot \mathbf{e_i^s}, \mathbf{1^s}, \mathbf{0^m})$ because there is no way to extend the value 2 in the i-th component of the first control section to 3 (since any period in $P' \cup P''$ adds either 0 or a value greater than 1 to this component). Given that we do not employ these vectors, it follows that any representation of $\mathbf{x}$ as a member of $\langle P' \cup P'' \rangle$ must be of the form $\mathbf{x} = \mathbf{x}' + \mathbf{x}''$ for some essential vector $\mathbf{x}' \in \langle P' \rangle$ and some vector $\mathbf{x}'' \in \langle P' \cup P'' \rangle$ (because, without employing an essential vector from $\langle P' \rangle$, we wouldn't get $\mathbf{1^s}$ into the second control section). Since $\mathbf{x}'$ is essential, it will already contribute $(\mathbf{3^s}, \mathbf{1^s}, \mathbf{1^m})$ to the three control sections. It follows that $\mathbf{x}'' = \mathbf{0^{2m+2s}}$ because adding any period from $P' \cup P''$ to $\mathbf{x}'$ will destroy the pattern $(\mathbf{3^s}, \mathbf{1^s}, \mathbf{1^m})$ in the control sections or will induce a component of value at least n in the base section (which is larger than any component of $\mathbf{x}$ in the base section). It follows that $\mathbf{x} = \mathbf{x}' \in \langle P' \rangle$.

It can be shown by standard arguments that the transformation $(E, E') \mapsto (\mathbf{c}, P, P', P'')$ is logspace-computable (even when numbers are encoded in unary). Finally observe that the above definition of essential vectors implies that every essential vector from $\mathbf{c} + \langle P \rangle$ employs a coefficient vector from $\{0, 1\}^{|P|}$ with precisely s ones. Since any inessential vector from $\mathbf{c} + \langle P \rangle$ also belongs to $\langle P' \rangle \subseteq \langle P' \cup P'' \rangle$, the promised condition (1) is satisfied (with $P' \cup P''$ at the place of Q). This concludes the proof. ◀

We will show in the sequel that the containment problem for 1-dimensional linear sets (with numerical input parameters given in binary representation) is log-hard in Π_2^p. To this end, we will make use of the following result on the aggregation of diophantine equations:

▶ **Lemma 11** ([3]). *Let*

$$\sum_{j=1}^{r} a_{1j}x_j = b_1 \quad \text{and} \quad \sum_{j=1}^{r} a_{2j}x_j = b_2 \tag{5}$$

be a system of two linear diophantine equations where a_{1j}, a_{2j} are non-negative integers and b_1, b_2 are strictly positive integers. Let t_1, t_2 be positive integers satisfying the following conditions:

1. *t_1 and t_2 are relatively prime.*
2. *t_1 does not divide b_2 and t_2 does not divide b_1.*
3. *$t_1 > b_2 - a_2$ and $t_2 > b_1 - a_1$ where a_i denotes the smallest nonzero coefficient in $\{a_{i1}, \ldots, a_{ir}\}$.*

Then, restricting x_j to non-negative integers, the solution set of (5) is the same as the solution set of

$$t_1 \cdot \sum_{j=1}^{r} a_{1j}x_j + t_2 \cdot \sum_{j=1}^{r} a_{2j}x_j = t_1 \cdot b_1 + t_2 \cdot b_2 \ .$$

Note that

$$t_1 = 1 + \max\{b_1, b_2\} \ \text{ and } t_2 = 1 + t_1 \tag{6}$$

is among the choices for t_1, t_2 such that the three conditions mentioned in Theorem 11 are satisfied.

From Lemma 11, the following result can be derived:

▶ **Lemma 12.** *Let $\mathbf{c} \in \mathbb{N}_0^m$, $A \in \mathbb{N}_0^{m \times r}$ and $A' \in \mathbb{N}_0^{m \times r'}$. Let $A_1, \ldots, A_m$ and $A'_1, \ldots, A'_m$ denote the row vectors of A and A', respectively. Let $s \geq 1$ and*

$$K_s = \{\mathbf{x} \in \{0,1\}^r : \sum_{i=1}^{r} x_i = s\} \ .$$

Suppose that the following holds:

$$\forall \mathbf{x} \in \mathbb{N}_0^r \setminus K_s, \exists \mathbf{y} \in \mathbb{N}_0^{r'} : \mathbf{c} + A\mathbf{x} = A'\mathbf{y} \ . \tag{7}$$

Then there exist $t_1^*, \ldots, t_m^* \in \mathbb{N}$ such that

$$(\forall \mathbf{x} \in \mathbb{N}_0^r, \exists \mathbf{y} \in \mathbb{N}_0^{r'} : \mathbf{c} + A\mathbf{x} = A'\mathbf{y}) \Leftrightarrow \left(\forall \mathbf{x} \in \mathbb{N}_0^r, \exists \mathbf{y} \in \mathbb{N}_0^{r'} : \sum_{j=1}^{m} t_j^* c_j + \sum_{j=1}^{m} t_j^* A_j \mathbf{x} = \sum_{j=1}^{m} t_j^* A'_j \mathbf{y} \right) \ . \tag{8}$$

Moreover, the aggregation coefficients $t_1^*, \ldots, t_m^*$ are logspace-computable from c, A, A'.

Proof. A solution for a system of m diophantine equations is always a solution for a single equation that represents an aggregation of the m given equations (regardless of how the aggregation coefficients $t_1^*, \ldots, t_m^*$ are chosen). Hence the equivalence (8) certainly holds for every $\mathbf{x} \in \mathbb{N}_0^r \setminus K_s$ and the direction "⇒" certainly holds for every $\mathbf{x} \in K_s$. Therefore, we need to verify only that there exist $t_1^*, \ldots, t_m^* \in \mathbb{N}$ such that the following implication is valid:

$$(\exists \mathbf{x} \in K_s, \forall \mathbf{y} \in \mathbb{N}_0^{r'} : \mathbf{c} + A\mathbf{x} \neq A'\mathbf{y}) \Rightarrow \left(\exists \mathbf{x} \in K_s, \forall \mathbf{y} \in \mathbb{N}_0^{r'} : \sum_{j=1}^{m} t_j^* c_j + \sum_{j=1}^{m} t_j^* A_j \mathbf{x} \neq \sum_{j=1}^{m} t_j^* A'_j \mathbf{y} \right) \ . \tag{9}$$

It is evident that (9) follows from (7) if A is the all-zeros matrix. We assume therefore in the sequel that A has at least one entry in $\mathbb{N}$. Clearly $\mathbf{c} + A\mathbf{x} = A'\mathbf{y}$ can be written in the form

$$\begin{bmatrix} -A & A' \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \mathbf{c} \ .$$

Moreover, for $\mathbf{x} \in K_s$ and any $u > 0$, it can be written as follows:

$$\begin{bmatrix} (uJ - A) & A' \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \mathbf{c} + u \cdot s \cdot \mathbf{1^m} \ . \tag{10}$$

Here J denotes the $m \times r$ all-ones matrix. Setting u equal to the largest absolute value of an entry in the matrix $-A$, the matrix $uJ - A$ has non-negative entries. Note that $u \geq 1$ since A has at least one entry in $\mathbb{N}$. Hence $\mathbf{c} + us\mathbf{1^m} \in \mathbb{N}^m$ so that we may bring Lemma 11 into play. Actually, we will apply this lemma iteratively in stages. In the first stage, we decompose the m diophantine equations in (10) into $m/2$ pairs, and we aggregate every pair into a single equation (by virtue of Lemma 11). After Stage 1, we are left with $m/2$ diophantine equations. Iterating this procedure for a total of $\lceil \log(m) \rceil$ stages, we finally arrive at a single diophantine equation whose solution space in $\mathbb{N}_0^{r+r'}$ coincides with the solution space for (10) in $\mathbb{N}_0^{r+r'}$. Moreover, for all $(\mathbf{x}, \mathbf{y}) \in K_s \times \mathbb{N}_0^{r'}$, it even coincides with the solution space for $\mathbf{c} + A\mathbf{x} = A'\mathbf{y}$. Hence the implication (9) is valid, as desired.
Since, in any individual application of Lemma 11, the coefficients t_1, t_2 can be chosen according to (6), the final aggregation coefficients $t_1^*, \ldots, t_m^*$ are easy to compute and, in fact, logspace computable from $\mathbf{c}, A, A'$ if all details are filled in properly. ◀

We are ready now for the next result:

▶ **Theorem 13.** *The containment problem for* 1*-dimensional linear sets is* log*-hard in* Π_2^p.

Proof. We reuse the notations from the proof of Theorem 10. Within that proof, we described a transformation $(E, E') \mapsto (\mathbf{c}, P, P', P'')$ which maps an instance of the containment problem for simple unary $(+, \cup)$-expressions into an instance of the containment problem for linear sets such that the latter satisfies the promised condition (1)[2] and such that the equivalence (3) is valid. Let d denote denote the dimension of the linear sets $\mathbf{c} + \langle P \rangle$ and $\langle P' \cup P'' \rangle$. Moreover let $r = |P|$ and $r' = |P' \cup P''|$. Let A be the $(d \times r)$-matrix with the periods from P as column vectors. Similarly, let A' be the $(d \times r')$- matrix with periods from $P' \cup P''$ as column vectors. It follows immediately from (3) that $L(E) \subseteq L(E')$ iff

$$\forall \mathbf{x} \in \mathbb{N}_0^r, \exists \mathbf{y} \in \mathbb{N}_0^{r'} : \mathbf{c} + A\mathbf{x} = A'\mathbf{y} \ .$$

Note that condition (1), written in matrix notation, translates into (7). According to Lemma 12, there exist $t_1^*, \ldots, t_m^*$ such that the equivalence in (8) is valid. Setting $c_0 = \sum_{j=1}^m t_j^* c_j$, $q_i = \sum_{j=1}^m t_j^* A_{ji}$ for $i = 1, \ldots, r$, and $q_i' = \sum_{j=1}^m t_j^* A'_{ji}$ for $i = 1, \ldots, r'$, $Q = \{q_1, \ldots, q_r\}$ and $Q' = \{q_1', \ldots, q_{r'}'\}$, we obtain a transformation $(E, E') \mapsto (c_0, Q, Q')$, which witnesses that the containment problem for 1-dimensional linear sets is log-hard in Π_2^p. ◀

Combining the restrictions of dimensionality 1 and unary encoding of numbers, the containment problem for linear sets becomes solvable in polynomial time:

▶ **Theorem 14.** *The containment problem for* 1*-dimensional linear sets with a unary encoding of numbers is in* P.

[2] with $P' \cup P''$ at the place of Q

Proof. Consider an input instance given by (the unary encoding of) c, P, c', P' with $c, c' \in \mathbb{N}_0$ and $P, P' \subset \mathbb{N}$. Let g (resp. g') be the greatest common divisor of the periods in P (resp. in P'). We make the following observation:

Claim: The containment $c + \langle P \rangle \subseteq c' + \langle P' \rangle$ is possible only if $c' \leq c$ and if g' is a divisor of g and of $c - c'$.

Given the assertion in the claim, we can accomplish the proof as follows. Setting $c_0 = c - c'$, our original question, "$c + \langle P \rangle \subseteq c' + \langle P' \rangle$?", is equivalent to "$c_0 + \langle P \rangle \subseteq \langle P' \rangle$?". We may now even assume that $g' = 1$ (because, if necessary, we can divide all numerical parameters by g'). If 1 is among the periods of P', then the answer to "$c_0 + \langle P \rangle \subseteq \langle P' \rangle$?" is clearly "yes". Suppose now that $1 \notin P'$. It is well known that $\langle P' \rangle$ contains all but finitely many natural numbers [6]. Let $F(P')$ (called the *Frobenius number* of P') denote the largest number in $\mathbb{N}$ that is not contained in $\langle P' \rangle$. It is well known that $F(P') < (\max(P') - 1) \cdot (\min(P') - 1)$ [1]. The questions "$x \in c_0 + \langle P \rangle$?" and "$x \in \langle P' \rangle$?" can be answered for all $x < (\max(P') - 1) \cdot (\min(P') - 1)$ in the obvious way by dynamic programming. Given the answers to these questions, we can immediately decide whether $c_0 + \langle P \rangle \subseteq \langle P' \rangle$.

All that remains to be done is proving the above claim. Suppose that

$$c + \langle P \rangle \subseteq c' + \langle P' \rangle \ . \tag{11}$$

This obviously implies that $c' \leq c$. It is furthermore obvious that $\langle P \rangle \subseteq g \cdot \mathbb{N}_0$ and $\langle P' \rangle \subseteq g' \cdot \mathbb{N}_0$. Moreover, by the definition of the Frobenius number, $s := g \cdot F\left((\frac{1}{g} \cdot \langle P \rangle)\right)$ is the largest multiple of g that does not belong to $\langle P \rangle$. Hence $c + s + g, c + s + 2g \in c + \langle P \rangle$ and, because of (11), there must exist $q_2 > q_1 \geq 1$ such that $c + s + g = c' + q_1 g'$ and $c + s + 2g = c' + q_2 g'$. Now we obtain $g = (q_2 - q_1)g'$ so that g' is a divisor of g. Since $(c - c') + \langle P \rangle \subseteq \langle P' \rangle \subseteq g' \cdot \mathbb{N}_0$ and $\langle P \rangle$ contains only multiples of g' (because it only contains multiples of g), it follows that g' must also be a divisor of $c - c'$, which concludes the proof of the claim and the proof of the theorem. ◀

4 Proof of Theorem 9

It is easy to see that the containment problem for simple unary $(+, \cup)$-expressions is a member of the complexity class Π_2^p. In somewhat more detail, let E and E' be two simple unary expressions of the form (2). Then $L(E) \subseteq L(E')$ iff

$$\forall a \in \{1,2\}^s, \exists a' \in \{1,2\}^{s'} : \sum_{i=1}^{s} \mathbf{B}_{\mathbf{i}\mathbf{a}_\mathbf{i}} = \sum_{i=1}^{s'} \mathbf{B}'_{\mathbf{i}\mathbf{a}'_\mathbf{i}} \ .$$

The membership in Π_2^p is now immediate from a well known characterization of Π_2^p due to Wrathall [8]: $L \in \Pi_2^p$ iff there exists a polynomial q and a language $L_0 \in \mathrm{P}$ such that

$$L = \{x | (\forall y_1 \text{ with } |y_1| \leq q(|x|))(\exists y_2 \text{ with } |y_2| \leq q(|x|)) : \langle y_1, y_2, x \rangle \in L_0\} \ .$$

It remains to show that it is log-hard in Π_2^p. To this end, we will design a logspace reduction from $\overline{\mathcal{B}_2}^{DNF}$ to this problem. Let $f(X_1, X_2)$ be an instance of $\overline{\mathcal{B}_2}^{DNF}$ (as described in Example 4). Since f employs only finitely many variables, we may assume that $X_i = \{x_{i1}, \ldots, x_{in}\}$ for $i = 1, 2$ and some $n \geq 1$. As a DNF-formula, f is the disjunction of Boolean monomials, say $f = M_1 \vee \ldots \vee M_m$. We may clearly assume that none of the monomials contains the same variable twice. We will transform $f(X_1, X_2)$ into simple unary $(+, \cup)$-expressions E_1 and E_2 such that

$$(\forall X_1, \exists X_2 : f(X_1, X_2) = 0) \Leftrightarrow (L(E_1) \subseteq L(E_2)) \ . \tag{12}$$

For all $i = 1, \ldots, n$ and $j = 1, \ldots, m$, let

$$\mathbf{b_{1i}}[j] = \begin{cases} 1 & \text{if } x_{1i} \in M_j \\ 0 & \text{otherwise} \end{cases} ,$$

i.e., the binary vector $\mathbf{b_{1i}} \in \{0,1\}^m$ indicates in which monomials the variable x_{1i} actually occurs. Let $\mathbf{b'_{1i}} \in \{0,1\}^m$ denote the corresponding vector with indicator bits for the occurrences of $\overline{x_{1i}}$ within $M_1, \ldots, M_m$. Let the vectors $\overline{\mathbf{b_{1i}}}$ and $\overline{\mathbf{b'_{1i}}}$ be obtained from $\mathbf{b_{1i}}$ and $\mathbf{b'_{1i}}$, respectively, by bitwise negation. Clearly, the bits of these vectors indicate the non-occurrences of x_{1i} resp. $\overline{x_{1i}}$ within $M_1, \ldots, M_m$. Let $\mathbf{b_{2i}}, \mathbf{b'_{2i}}, \overline{\mathbf{b_{2i}}}, \overline{\mathbf{b'_{2i}}}$ be the corresponding vectors with indicator bits for the occurrences resp. non-occurrences of the variable x_{2i}. We now define a couple of $(+, \cup)$-expressions:

$$E'_1 = \sum_{i=1}^{n} (\mathbf{1^m} \cup \mathbf{1^m}) \quad \text{and} \quad E_1 = E'_1 + \sum_{i=1}^{n} (\overline{\mathbf{b_{1i}}} \cup \overline{\mathbf{b'_{1i}}})$$

$$E'_2 = \sum_{j=1}^{m} \sum_{i=1}^{2n-1} (\mathbf{e_j^m} \cup \mathbf{0^m}) \quad \text{and} \quad E_2 = E'_2 + \sum_{i=1}^{n} (\mathbf{b_{2i}} \cup \mathbf{b'_{2i}}) \ .$$

The following immediate observations will prove useful:

1. $L(E'_1) = \{n \cdot \mathbf{1^m}\}$ and $L(E'_2) = \{0, \ldots, 2n-1\}^m$.
2. $L(E_1) \subseteq \{n, \ldots, 2n\}^m$ and $L(E_2) \supseteq \{n, \ldots, 2n-1\}^m$.

Note that the only vectors of $L(E_1)$ which might perhaps not belong to $L(E_2)$ are the ones with at least one component of size $2n$. The following definitions take care of these "critical vectors". We say that a partial assignment of the variables in $X_1 \cup X_2$ *annuls* M_j if one of the literals contained in M_j is set to 0. Let $\mathbf{y} \in \{n, \ldots, 2n\}^m$. An assignment $A_1 : X_1 \to \{0,1\}$ is said to be an X_1-*assignment of type* $\mathbf{y}$ if the following holds:

$$\forall j = 1, \ldots, m : (\mathbf{y}[j] = 2n \Leftrightarrow A_1 \text{ does not annul } M_j) \ .$$

We say that $A_2 : X_2 \to \{0,1\}$ is an X_2-*assignment of type* $\mathbf{y}$ if the following holds:

$$\forall j = 1, \ldots, m : (\mathbf{y}[j] = 2n \Rightarrow A_2 \text{ annuls } M_j) \ .$$

The desired equivalence (12) is easy to derive from the following claims:

Claim 1: For every $\mathbf{y} \in L(E_1)$, there exists an X_1-assignment A_1 of type $\mathbf{y}$.

Claim 2: For every $A_1 : X_1 \to \{0,1\}$, there exists $\mathbf{y} \in L(E_1)$ such that A_1 is an X_1-assignment of type $\mathbf{y}$.

Claim 3: For every $\mathbf{y} \in \{n, \ldots, 2n\}^m$:

$$\mathbf{y} \in L(E_2) \Leftrightarrow (\exists A_2 : X_2 \to \{0,1\} : A_2 \text{ is an } X_2\text{-assignment of type } \mathbf{y}) \ .$$

Proof of Claim 1: Pick any $\mathbf{y} \in L(E_1)$. It follows that $\mathbf{y}$ is of the form

$$\mathbf{y} = n \cdot \mathbf{1^m} + \sum_{i=1}^{n} \widetilde{\mathbf{b_{1i}}} \quad \text{with} \quad \widetilde{\mathbf{b_{1i}}} \in \{\overline{\mathbf{b_{1i}}}, \overline{\mathbf{b'_{1i}}}\} \ . \tag{13}$$

If $\widetilde{\mathbf{b_{1i}}} = \overline{\mathbf{b_{1i}}}$, we set $A_1(x_{1i}) = 0$ else, if $\widetilde{\mathbf{b_{1i}}} = \overline{\mathbf{b'_{1i}}}$, we set $A_1(x_{1i}) = 1$. We claim that A_1 is of type $\mathbf{y}$. This can be seen as follows. Pick any $j \in \{1, \ldots, m\}$. An inspection of (13) reveals the following:

- Suppose that $\mathbf{y}[j] = 2n$. It follows that $\widetilde{\mathbf{b_{1i}}}[j] = 1$ for $i = 1, \ldots, n$. Hence, if $\widetilde{\mathbf{b_{1i}}} = \overline{\mathbf{b_{1i}}}$, then $A_1(x_{1i}) = 0$, $\overline{\mathbf{b_{1i}}}[j] = 1$ and, therefore, $x_{1i} \notin M_j$. Similarly, if $\widetilde{\mathbf{b_{1i}}} = \overline{\mathbf{b'_{1i}}}$, then $A_1(x_{1i}) = 1$, $\overline{\mathbf{b'_{1i}}}[j] = 1$ and, therefore, $\overline{x_{1i}} \notin M_j$. Since these observations hold for all $i = 1, \ldots, n$, we may conclude that A_1 does not annul M_j.

- Suppose that $\mathbf{y}[j] \leq 2n-1$. Then there exists $i \in \{1,\ldots,n\}$ such that $\widetilde{\mathbf{b_{1i}}}[j] = 0$. Hence, if $\widetilde{\mathbf{b_{1i}}} = \overline{\mathbf{b_{1i}}}$, then $A_1(x_{1i}) = 0$, $\overline{\mathbf{b_{1i}}}[j] = 0$ and, therefore, $x_{1i} \in M_j$. Similarly, if $\widetilde{\mathbf{b_{1i}}} = \overline{\mathbf{b'_{1i}}}$, then $A_1(x_{1i}) = 1$, $\overline{\mathbf{b'_{1i}}}[j] = 0$ and, therefore, $\overline{x_{1i}} \in M_j$. It follows that A_1 does annul M_j.

The above discussion shows that A_1 is of type $\mathbf{y}$, indeed.

Proof of Claim 2: Given any $A_1 : X_1 \to \{0,1\}$, we set $\mathbf{y} = n \cdot \mathbf{1^m} + \sum_{i=1}^{n} \widetilde{\mathbf{b_{1i}}}$ where $\widetilde{\mathbf{b_{1i}}} = \overline{\mathbf{b_{1i}}}$ if $A_1(x_{1i}) = 0$ and, similarly, $\widetilde{\mathbf{b_{1i}}} = \overline{\mathbf{b'_{1i}}}$ if $A_1(x_{1i}) = 1$. Note that, with this definition of $\mathbf{y}$, A_1 is precisely the X_1-assignment that we had chosen in the proof of Claim 1. As argued in the proof of Claim 1 already, A_1 is of type $\mathbf{y}$.

Proof of Claim 3: Pick any $\mathbf{y} \in \{n,\ldots,2n\}^m$. Suppose first that $\mathbf{y} \in L(E_2)$. It follows that $\mathbf{y}$ is of the form

$$\mathbf{y} = \mathbf{y}' + \sum_{i=1}^{n} \widetilde{\mathbf{b_{2i}}} \quad \text{with} \quad \mathbf{y}' \in \{0,\ldots,2n-1\}^m \quad \text{and} \quad \widetilde{\mathbf{b_{2i}}} \in \{\mathbf{b_{2i}}, \mathbf{b'_{2i}}\} \ . \tag{14}$$

If $\widetilde{\mathbf{b_{2i}}} = \mathbf{b_{2i}}$, we set $A_2(x_{2i}) = 0$ else, if $\widetilde{\mathbf{b_{2i}}} = \mathbf{b'_{2i}}$, we set $A_2(x_{2i}) = 1$. We claim that A_2 is of type $\mathbf{y}$. Consider an index $j \in \{1,\ldots,m\}$ such that $\mathbf{y}[j] = 2n$. An inspection of (14) reveals that there exists $i \in \{1,\ldots,n\}$ such that $\widetilde{\mathbf{b_{2i}}}[j] = 1$. If $\widetilde{\mathbf{b_{2i}}} = \mathbf{b_{2i}}$, then $A_2(x_{2i}) = 0$, $\mathbf{b_{2i}}[j] = 1$ and, therefore, $x_{2i} \in M_j$. Similarly, if $\mathbf{b_{2i}} = \mathbf{b'_{2i}}$, then $A_2(x_{2i}) = 1$, $\mathbf{b'_{2i}}[j] = 1$ and, therefore, $\overline{x_{2i}} \in M_j$. In any case, A_2 annuls M_j and we may conclude that A_2 is of type $\mathbf{y}$.

Suppose now that there exists an X_2-assignment A_2 that is of type $\mathbf{y} \in \{n,\ldots,2n\}^m$. We define $\mathbf{y}'' = \sum_{i=1}^{n} \widetilde{\mathbf{b_{2i}}}$ where $\widetilde{\mathbf{b_{2i}}} = \mathbf{b_{2i}}$ if $A_2(x_{2i}) = 0$ and, similarly, $\widetilde{\mathbf{b_{2i}}} = \mathbf{b'_{2i}}$ if $A_2(x_{2i}) = 1$. Since A_2 is of type $\mathbf{y}$, it annuls every M_j with $\mathbf{y}[j] = 2n$. It follows that, for every $j \in \{1,\ldots,m\}$ with $\mathbf{y}[j] = 2n$, there exists $i \in \{1,\ldots,n\}$ such either $x_{2i} \in M_j$ and $A_2(x_{2i}) = 0$ or $\overline{x_{2i}} \in M_j$ and $A_2(x_{2i}) = 1$. In both cases, we have that $\widetilde{\mathbf{b_{2i}}}[j] = 1$. It follows from this discussion that $\mathbf{y}''[j] \geq 1$ for every j with $\mathbf{y}[j] = 2n$. Obviously $\mathbf{y}''[j] \leq n$ for all $j = 1,\ldots,m$. Since $L(E'_2) = \{0,\ldots,2n-1\}^m$ and $\mathbf{y} \in \{n,\ldots,2n\}^m$, there exists $\mathbf{y}' \in L(E_2)$ such that $\mathbf{y} = \mathbf{y}' + \mathbf{y}''$. This decomposition of $\mathbf{y}$ shows that $\mathbf{y} \in L(E_2)$.

We are ready now for proving (12). Assume first that the condition on the left hand-side of (12) is valid. Pick any $\mathbf{y} \in L(E_1)$. Pick an X_1-assignment A_1 of type $\mathbf{y}$ (application of Claim 1). It follows that the monomials M_j with $\mathbf{y}[j] = 2n$ are not yet annulled by A_1. According to the left hand-side of (12), there must exist an assignment $A_2 : X_2 \to \{0,1\}$ that annuls them. In other words: A_2 is an X_2-assignment of type $\mathbf{y}$. We may now conclude from Claim 3 that $\mathbf{y} \in L(E_2)$, as desired.

Suppose now that $L(E_1) \subseteq L(E_2)$. Pick any assignment $A_1 : X_1 \to \{0,1\}$. Pick $\mathbf{y} \in L(E_1)$ such A_1 is an X_1-assignment of type $\mathbf{y}$ (application of Claim 2). It follows that only the monomials M_j with $\mathbf{y}[j] = 2n$ are not yet annulled by A_1. Since $\mathbf{y}$, as an element of $L(E_1)$, must satisfy $\mathbf{y} \in \{n,\ldots,2n\}^m$ and must furthermore belong to $L(E_2)$, we may conclude from Claim 3 that there exists an X_2-assignment $A_2 : X_2 \to \{0,1\}$ of type $\mathbf{y}$. In other words: A_2 annuls all monomials M_j with $\mathbf{y}[j] = 2n$. It follows from this discussion that the condition on the left hand-side of (12) is valid, which concludes the proof.

5 Open Problems

In the proof of our hardness results, we made essential use of the fact that $\langle P \rangle$ contains all linear combinations of the periods in P with coefficient vectors from $\mathbb{N}_0^{|P|}$. We would be interested to know whether the computational complexity of the containment problem is still the same when we deal with coefficient vectors from $\mathbb{N}^{|P|}$ (thereby ruling out 0-coefficients).

References

1 Alfred Brauer. On a problem of partitions. *American Journal of Mathematics*, 64(1):299–312, 1942.

2 Dmitry Chistikov, Christoph Haase, and Simon Halfon. Context-free commutative grammars with integer counters and resets. *Theoretical Computer Science*, 2016. In press. Online version: https://doi.org/10.1016/j.tcs.2016.06.017.

3 F. Glover and R. E. D. Woolsey. Aggregating diophantine equations. *Zeitschrift für Operations Research*, 16:1–10, 1972.

4 Thiet-Dung Huynh. The complexity of semilinear sets. *Elektronische Informationsverarbeitung und Kybernetik*, 18(6):291–338, 1982.

5 Rohit J. Parikh. On context-free languages. *Journal of the Association on Computing Machinery*, 13(4):570–581, 1966.

6 José C. Rosales and Pedro A. García-Sánchez. *Numerical Semigroups*. Springer, 2009.

7 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1977.

8 Celia Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.

Improving the Upper Bound on the Length of the Shortest Reset Words

Marek Szykuła
Institute of Computer Science,
University of Wrocław, Wrocław, Poland
msz@cs.uni.wroc.pl

Abstract

We improve the best known upper bound on the length of the shortest reset words of synchronizing automata. The new bound is slightly better than $114n^3/685+O(n^2)$. The Černý conjecture states that $(n-1)^2$ is an upper bound. So far, the best general upper bound was $(n^3-n)/6-1$ obtained by J.-E. Pin and P. Frankl in 1982. Despite a number of efforts, it remained unchanged for about 35 years.

To obtain the new upper bound we utilize avoiding words. A word is avoiding for a state q if after reading the word the automaton cannot be in q. We obtain upper bounds on the length of the shortest avoiding words, and using the approach of Trahtman from 2011 combined with the well-known Frankl theorem from 1982, we improve the general upper bound on the length of the shortest reset words. For all the bounds, there exist polynomial algorithms finding a word of length not exceeding the bound.

2012 ACM Subject Classification Mathematics of computing → Combinatoric problems, Theory of computation → Formal languages and automata theory

Keywords and phrases Avoiding Word, Černý Conjecture, Reset Length, Reset Threshold, Synchronizing Automaton

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.56

Funding Supported in part by the National Science Centre, Poland under project number 2017/25/B/ST6/01920.

Acknowledgements I thank Mikhail Berlinkov, Costanza Catalano, Vladimir Gusev, Jakub Kośmider, Jakub Kowalski, and the anonymous reviewers for proofreading and useful comments.

1 Introduction

We deal with deterministic finite complete (semi)automata $\mathscr{A}(Q,\Sigma,\delta)$, where Q is the set of *states*, Σ is the input *alphabet*, and $\delta\colon Q\times\Sigma\to Q$ is the *transition function*. We extend δ to the function $Q\times\Sigma^*\to Q$ in the usual way. Throughout the paper, by n we denote the number of states $|Q|$.

By $\Sigma^{\leq i}$ we denote the set of all words over Σ of length at most i. Given a state $q\in Q$ and a word $w\in\Sigma^*$ we write shortly $q\cdot w=\delta(q,w)$. Given a subset $S\subseteq Q$ we write $S\cdot w$ for the image $\{q\cdot w \mid q\in S\}$. Then, $S\cdot w^{-1}$ is the preimage $\{q\in Q \mid q\cdot w\in S\}$, and when S is a singleton we also write $q\cdot w^{-1}=\{q\}\cdot w^{-1}$.

The *rank* of a word $w\in\Sigma^*$ is the cardinality of the image of Q under the action of this word: $|Q\cdot w|$. A word is *reset* or *synchronizing* if it has rank 1. An automaton is *synchronizing* if it admits a reset word. The *reset threshold* $\mathrm{rt}(\mathscr{A})$ is the length of the shortest reset words.

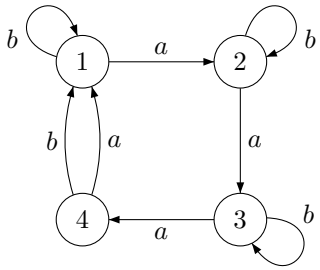


Figure 1 The Černý automaton with 4 states.

We say that a word $w \in \Sigma^*$ *compresses* a subset $S \subseteq Q$ if $|S \cdot w| < |S|$. A word $w \in \Sigma^*$ *avoids* a state $q \in Q$ if $q \notin Q \cdot w$. A state that admits an avoiding word is *avoidable*. We also say that a state q is *avoidable from a subset* S if there exists a word w such that $q \notin S \cdot w$.

The famous Černý conjecture, formally formulated in 1969, is one of the most longstanding open problems in automata theory. It states that every synchronizing n-state automaton has a reset word of length at most $(n-1)^2$. This bound would be tight, since it is reached for every n by the Černý automata [7]. Fig. 1 shows the Černý automaton with $n = 4$ states. Its shortest reset word is ba^3ba^3b.

The first general upper bound for the reset threshold given by Černý in [7] was $2^n - n - 1$. Later, it was improved several times: $\frac{1}{2}n^3 - \frac{3}{2}n^2 + n + 1$ given by Starke [23] in 1966, $\frac{1}{3}n^3 - \frac{3}{2}n^2 + 25/6n - 4$ by Černý, Pirická, and Rosenauerová [8] in 1971, $\frac{7}{27}n^3 - 17/18n^2 + 17/6n - 3$ by Pin [19] in 1978, and $(\frac{1}{2} - \frac{\pi}{36})n^3 + o(n^3)$ by Pin [21] in 1981.

Then, the well known upper bound was established in 1982 by Pin and Frankl through the following combinatorial theorem:

▶ **Theorem 1** ([12, 21])**.** *Let $\mathscr{A}(Q, \Sigma, \delta)$ be a strongly connected synchronizing automaton, and consider a subset $S \subseteq Q$ of cardinality ≥ 2. Then there exists a word such that $|S \cdot w| < |S|$ of length at most*

$$\frac{(n - |S| + 2) \cdot (n - |S| + 1)}{2}.$$

For integers $1 \leq i, j \leq n$ we define

$$C(j, i) \quad = \quad \sum_{s=i+1}^{j} \frac{(n - s + 2) \cdot (n - s + 1)}{2}.$$

From Theorem 1, $C(j, i)$ is an upper bound on the length of the shortest words compressing a subset of size j to a subset of size at most i: starting from a subset S of size j, we iteratively apply Theorem 1 to bound the length of a shortest word compressing each (in the worst case) of the obtained subsets of sizes $j, j-1, \ldots, i+1$. This yields the well known bound on the length of the shortest reset words:

$$\mathrm{rt}(\mathscr{A}) \leq C(n, 1) = \frac{n^3 - n}{6}.$$

This bound was also discovered independently in [17]. Actually, the best bound was $\frac{n^3-n}{6} - 1$ (for $n \geq 4$), since Pin [21] proved that (for $n \geq 4$) there is a word compressing Q to a subset of size $n - 3$ by a word of length 9 (instead of 10). Theorem 1 also bounds the lengths of a compressing word found by a greedy algorithm (e.g. [1, 11]), which is an algorithm finding a reset word by iterative application of a shortest word compressing the current subset. For about 35 years, there was no progress in improving the bound in the general case.

However, better bounds have been obtained for a lot of special classes of automata, for example for oriented (monotonic) automata [11], circular automata [10], Eulerian automata [15], aperiodic automata [26], generalized and weakly monotonic automata [2, 29], automata with a sink (zero) state [18], one-cluster automata [3, 25], quasi-Eulerian and quasi-one-cluster automata [5], automata respecting intervals of a directed graph [14], decoders of finite prefix codes [4, 6], automata with a letter of small rank [4, 20], and 1-contracting automata [9]. See also [28] for a survey.

In 2011, Trahtman claimed the better upper bound $(7n^3+6n-16)/48$ [27]. Unfortunately, the proof contains an error, and so the result remains unproved. The idea was to utilize avoiding words; [27, Lemma 3] states that for every $q \in Q$ there exists an avoiding word of length at most $n-1$. A counterexample to this was found in [13], where it was also suggested that providing any linear upper bound on the length of avoiding words would also imply an improvement for the upper bound on the reset threshold.

The avoiding word problem is similar to synchronization: instead of bringing the automaton into one state, we ask how long word we require to not being in a particular state. For the automaton from Fig. 1, the shortest avoiding words for states 1, 2, 3, 4 are ba, baa, $baaa$, and b, respectively. So far, only a trivial cubic upper bound $\mathrm{rt}(\mathscr{A})+1$ was known for synchronizing automata. Avoiding words do not necessarily exist in general, but they always do for every state in the case of a synchronizing automaton unless there is a *sink state* ([18]), for which all letters act like identity.

The main contributions in this paper are as follows: We prove upper bounds on the length of the shortest avoiding words, in particular the quadratic bound $(n-1)(n-2)+2$. Also, the length of avoiding words is connected with the length of compressing words. We show that for every state q and a subset of states S, either there is a short avoiding word for q from S or a short compressing word for S. This connection leads to the main idea for the improvement of the general upper bound on the reset threshold: either improve by avoiding words, or use shorter compressing words directly to reduce the bound obtained by Theorem 1. In contrast to the previous approaches, which bounded the length of the compressing words independently for each size $|S|$, the new bound utilizes a conditional approach.

The new upper bound is

$$(85059n^3 + 90024n^2 + 196504n - 10648)/511104,$$

which is slightly better than the much simpler formula $114n^3/685 + O(n^2)$. The latter improves the coefficient of n^3 by $1/4110$. In the last section we discuss open problems and further possibilities for improvements.

2 Avoiding words

For the next lemma, we need to introduce a few definitions from linear algebra for automata (see, e.g., [4, 15, 20]). By $\mathbb{R}^n$ we denote the real n-dimensional linear space of row vectors. Without loss of generality we assume that $Q = \{1, 2, \ldots, n\}$. For a vector $v \in \mathbb{R}^n$, we denote the value at an i-th position by $v(i)$. For a subset $S \subseteq Q$, by $[S]$ we denote its characteristic row vector, which has $[S](i) = 1$ if $i \in S$, and $[S](i) = 0$ otherwise. Similarly, for a matrix M, we denote the value at an i-th row and a j-th column by $M(i, j)$. For a word $w \in \Sigma^*$, by $[w]$ we denote the $n \times n$ matrix of the transformation of w: $[w](i, j) = 1$ if $i \cdot w = j$ (state i is mapped to state j by the transformation of w), and $[w](i, j) = 0$ otherwise.

Right matrix multiplication corresponds to concatenation of two words; i.e. for every two words $u, v \in \Sigma^*$ we have $[uv] = [u] \cdot [v]$. For a subset S we have $([S][u])(i)$ equal to the number of states from S mapped by the transformation of u to state i. In particular,

$([S][u])(i) \geq 1$ if and only if $[S \cdot u](i) = 1$. Note that for $w \in \Sigma^*$, the matrix $[w]$ contains exactly one 1 in each row. Therefore, these are *stochastic* matrices, and we have the property that for any $v \in \mathbb{R}^n$, right matrix multiplication by $[w]$ preserves the sum of the entries, i.e. $\sum_{i \in Q}[v](i) = \sum_{i \in Q}([v][w])(i)$.

For example, for the automaton from Fig. 1 we have:

$$[a] = \begin{pmatrix} 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \\ 1&0&0&0 \end{pmatrix}, \ [b] = \begin{pmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 1&0&0&0 \end{pmatrix}, \ [ba] = \begin{pmatrix} 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \\ 0&1&0&0 \end{pmatrix}.$$

If $[S] = [1, 0, 1, 1]$, then $[S][ba] = [S][b][a] = [0, 2, 0, 1]$.

The linear subspace *spanned* by a set of vectors V is denoted by $\mathrm{span}(V)$. Given a linear subspace $L \subseteq \mathbb{R}^n$ and an $n \times n$ matrix m, the linear subspace mapped by m is $Lm = \{vm \mid v \in L\}$. The *dimension* of a linear subspace L is denoted by $\dim(L)$.

The following key lemma states that by a short (linear) word we can either avoid a state (or one of the states from some set A) from the current subset or compress the current subset.

▶ **Lemma 2.** *Let $\mathscr{A}(Q, \Sigma, \delta)$ be an n-state automaton. Consider a non-empty subset $S \subseteq Q$ and a non-empty proper subset $A \subsetneq S$. Suppose that there is a word $w \in \Sigma^*$ such that $A \nsubseteq S \cdot w$. Then there exists a word w length at most $n - |A|$ satisfying either*

1. *$A \nsubseteq S \cdot w$, or*
2. *$|S \cdot w| < |S|$.*

Proof. Let $L_i = \mathrm{span}(\{[S][w] \mid w \in \Sigma^{\leq i}\})$. We consider the following sequence of linear subspaces:

$$L_0 \subseteq L_1 \subseteq L_2 \subseteq \ldots,$$

and use the ascending chain condition (see, e.g., [4, 15, 20, 24]):

- If $L_k = L_{k+1}$, then we claim that also $L_{k+1} = L_{k+2} = \ldots$ holds. Observe that for all $i \geq 0$ we have:

$$L_{i+1} = \mathrm{span}\left(L_i \cup \bigcup_{a \in \Sigma} L_i[a]\right).$$

  Hence, if $L_k = L_{k+1}$, then for $i = k$ we obtain

$$L_{k+1} = \mathrm{span}\left(L_{k+1} \cup \bigcup_{a \in \Sigma} L_{k+1}[a]\right) = L_{k+2},$$

  and so $L_{k+i} = L_k$ for all $i \geq 0$.
- Let i be the smallest integer such that $L_i = L_{i+1}$. Then $m = \dim(L_i)$ is the maximum among the dimensions of the subspaces from the above sequence.
- $\dim(L_0) = 1$ and the dimensions grow by at least 1 up to m. Hence, we have

$$\dim(L_{n-|A|}) \geq \min\{m, n - |A| + 1\}.$$

Note that if for a word w the vector $v = [S][w]$ has $v(q) = 0$ for some $q \in A$, then $q \notin S \cdot w$, and we have Case (1). If $v = [S][w]$ has $v(q) \geq 2$ for some $q \in A$, then a pair of states from S is compressed by the action of w (to state q), and we have Case (2).

Now, we show that in the spanning set of $L_{n-|A|}$ there must be a vector that contains either 0 or an integer ≥ 2 at the position corresponding to a state from A, which implies that there exists a word w of length at most $n - |A|$ satisfying either Case (1) or Case (2). Suppose

for a contradiction that this is not the case. Every vector $v \in L_k$ is a linear combination of the vectors from the spanning set; let c be the sum of the coefficients of the spanning vectors in such a linear combination. Every vector $[S][w]$ in the spanning set has the sum of elements equal to $|S|$ and has 1 at all the positions corresponding to the states from A. Hence, the sum of the entries in v is equal to $c|S|$, and at every position corresponding to the states from A we have value c. The sum of the entries at the positions corresponding to the states from $Q \setminus A$ equals $c(|S| - |A|)$. Therefore, every $q \in A$ satisfies the following equality:

$$v(q) = \frac{1}{|S| - |A|} \cdot \sum_{p \in Q \setminus A} v(p).$$

It follows that the values at the positions corresponding to the states from A are completely determined by the sum of the values from the other positions, which means that the dimension of $L_{n-|A|}$ is at most $n - |A|$. We assumed in the lemma that there exists a word w avoiding a state from A. Hence, $[S][w]$ has 0 at some position corresponding to a state from A, and therefore breaks the above equality for this state, as the right side is non-zero. Therefore, the subspace $L_{|w|}$ must have a larger dimension than $\dim(L_{n-|A|})$. This means that the dimension of $L_{n-|A|}$ is not maximal, which contradicts $\dim(L_{n-|A|}) \geq \min\{m, n - |A| + 1\}$. ◀

Lemma 2 can be applied iteratively to obtain a word compressing the given subset to the desired size.

▶ **Lemma 3.** *Let $\mathscr{A}(Q, \Sigma, \delta)$ be an n-state automaton. Consider a non-empty subset $S \subseteq Q$ and a non-empty proper subset $A \subsetneq S$. Let $k \geq 1$ be an integer. Suppose that there exists a word $w \in \Sigma^*$ such that $A \not\subseteq S \cdot w$. Then there is a word w of length at most $k(n - |A|)$ satisfying either:*

1. *$A \not\subseteq S \cdot w$, or*
2. *$|S \cdot w| \leq |S| - k$.*

Proof. If Case (1) holds for some $w \in \Sigma^{\leq k(n-|A|)}$ then we are done; suppose this is not the case.

We iteratively apply Lemma 2 k times for subset A starting from subset S: For $i = 1, \ldots, k$ we apply the lemma for the subset $S \cdot w_1 \ldots w_{i-1}$, where $w_j \in \Sigma^{\leq n-|A|}$ is the word obtained from the lemma in the j-th iteration.

In every iteration, we must get Case (2) of Lemma 2 ($|S \cdot w| < |S|$), as otherwise $A \not\subseteq S \cdot w_1 \ldots w_i$, which contradicts our assumption that Case (1) does not hold for every word of length at most $k(n - |A|) \geq i(n - |A|)$. Also, for $i \leq k - 1$, we must have $A \subset S \cdot w_1 \ldots w_i$ (i.e. A is a proper subset); otherwise $A \not\subseteq S \cdot w_1 \ldots w_i a$ for some letter $a \in \Sigma$ as A contains a state that can be avoided from S, and this word has length at most $k(n - |A|)$ which again contradicts our assumption. Therefore, the conditions are met for every iteration so we can apply the lemma k times.

It follows that the obtained word $w_1 \ldots w_k$ is such that $|S \cdot w_1 \ldots w_k| \leq |S| - k$. ◀

If the subset A of states to avoid is large, the following approach can lead to a better bound:

▶ **Lemma 4.** *Let $\mathscr{A}(Q, \Sigma, \delta)$ be an n-state automaton. Consider a non-empty subset $S \subseteq Q$ and a non-empty subset $A \subseteq S$. If there exists a word $w \in \Sigma^*$ such that $A \not\subseteq S \cdot w$, then there exists such a word of length at most $(|S| - |A|)(n - |A|) + 1$.*

Proof. As in the proof of Lemma 3, we iteratively apply Lemma 2 at most $|S| - |A|$ times for subset A starting from subset S, stopping if the conditions are not met. It is possible that we do not do any iteration, which is the case when $A = S$.

In every iteration, we obtain a word w_i of length at most $n - |A|$. If we get $A \nsubseteq S \cdot w_1 \dots w_i$ in some i-th iteration, then we are done as the word $w_1 \dots w_i$ has length at most $(|S| - |A|)(n - |A|)$.

If we get $A = S \cdot w_1 \dots w_i$ for some $i \in \{0, \dots, |S| - |A|\}$, then observe that there must exist a letter $a \in \Sigma$ such that $A \cdot a \neq A$, because A contains an avoidable state from $S \supseteq A$. Note that since $|S \cdot w_1 \dots w_i| < |S \cdot w_1 \dots w_{i-1}|$ for every $i = 1, \dots, k$, after the $(|S| - |A|)$-th iteration we must have $|S \cdot w_1 \dots w_k| \leq |S| - (|S| - |A|) = |A|$, we must get this case after the last iteration. It follows that in any case we obtain the word $w_1 \dots w_i a$ of length at most $(|S| - |A|)(n - |A|) + 1$. ◀

We state a quadratic upper bound on the length of the shortest avoiding words:

▶ **Corollary 5.** *For $n \geq 2$, in an n-state automaton $\mathscr{A}(Q, \Sigma, \delta)$, for every non-empty proper subset $A \subset Q$ containing an avoidable state, there exists a word avoiding a state from A of length at most*

$$(n - 1 - |A|)(n - |A|) + 2.$$

Proof. Since there exists an avoidable state in A, there is a letter $a \in \Sigma$ such that $|Q \cdot a| < n$.

If $A \nsubseteq Q \cdot a$ then we are done with a word of length 1. Otherwise $A \subseteq Q \cdot a$, so we use Lemma 4 with subset A and subset $S = Q \cdot a$. Since there exists a word avoiding a state from A, the lemma yields a word w of length at most $(|S| - |A|)(n - |A|) + 1 \leq (n - 1 - |A|)(n - |A|) + 1$. Thus, aw avoids a state from A and has length at most $(n - 1 - |A|)(n - |A|) + 2$. ◀

In particular, we obtain the upper bound $(n - 2)(n - 1) + 2$ on the length of the shortest avoiding words for any state ($|A| = 1$).

▶ **Theorem 6.** *The words from Lemma 2, Lemma 3, Lemma 4, and Corollary 5 can be found in polynomial time.*

Proof. We use the reduction procedure from [4], which in polynomial time replaces each set $\Sigma^{\leq i}$ in the proof of Lemma 2 with a set W_i containing at most $i + 1$ words such that L_i has the same dimension.

The procedure starts for $i = 0$ with $\{\varepsilon\}$ (the set with the empty word) and inductively constructs a set W_i assuming we have found W_{i-1}. This is done by considering all words wa for $w \in W_{i-1}$ and $a \in \Sigma$ and setting $W_i = W_{i-1} \cup \{wa\}$ for which the dimension of the corresponding subspace grows. There always exists such a word wa, which is argued by the ascending chain condition.

Then, the set W_m is used to span the first linear subspace with the maximal dimension (L_m), so we can find a word satisfying Case (1) or Case (2) of Lemma 2 in W_m. It is obvious that the corresponding words from the other proofs are constructible in polynomial time. ◀

3 Improved bound on reset threshold

In this section, we consider a synchronizing n-state automaton $\mathscr{A}(Q, \Sigma, \delta)$. Obviously, in such an automaton, every state is avoidable unless there is a sink state (a state q such that $q \cdot a = q$ for all $a \in \Sigma$), which cannot be avoided. For synchronizing automata with a sink state the tight upper bound is $n(n - 1)/2$ (see, e.g., [22]). Thus we can assume that $\mathscr{A}$ does not have a sink state, and so Lemma 2 and Lemma 3 can be applied for every non-empty subset A.

▶ **Lemma 7.** *Let $w \in \Sigma^*$ and let $g = \min\{|q \cdot w^{-1}| \mid q \in Q \cdot w\}$. There are at least $(g+1)|Q \cdot w| - n$ states $q \in Q \cdot w$ such that $|q \cdot w^{-1}| = g$.*

Proof. Let d be the number of states $q \in Q \cdot w$ whose preimages under w^{-1} have size equal to g. So $|Q \cdot w| - d$ states have the preimages of size at least $g+1$. Note that $(Q \cdot w) \cdot w^{-1} = Q$, and that the sets $q \cdot w^{-1}$ and $p \cdot w^{-1}$ are disjoint for all pairs of states $q \neq p$. So $Q \cdot w^{-1}$ has cardinality at least $dg + (g+1)(|Q \cdot w| - d) = (g+1)|Q \cdot w| - d$. Since this cannot be larger than $n = |Q|$, we get $d \geq (g+1)|Q \cdot w| - n$. ◀

From Lemma 7, in particular, we get that there are at least $2|Q \cdot w| - n$ states in the image $Q \cdot w$ with a unique state in the preimage.

The following lemma is based on [27, Lemma 4], but with a more general bound:

▶ **Lemma 8.** *Let $w \in \Sigma^*$ be a word of rank $r \geq \lfloor (n+1)/2 \rfloor$. Suppose that for some integer $k \geq 1$, for every $A \subset Q$ of size $1 \leq |A| \leq n-1$, there is a word $v_A \in \Sigma^{\leq k(n-|A|)}$ such that $A \not\subseteq Q \cdot v_A$. Then there is a word of rank at most $n/2$ and length at most*

$$|w| + k\frac{n^2 - (2n - 2r - 1)^2}{4}.$$

Proof. For $i = r, r-1, \ldots, \lfloor n/2 \rfloor$, we inductively construct words w_i of length $\leq |w| + k(r-i)(2n-r-i-1)$ of rank at most i. First, let $w_r = w$.

Let $i < r$ and suppose that we have already found w_{i+1}. If already $|Q \cdot w_{i+1}| \leq i$ then we just set $w_i = w_{i+1}$. Otherwise, we have $|Q \cdot w_{i+1}| = i+1$.

Because $i + 1 \geq (n+1)/2$, there exists a non-empty subset of $Q \cdot w_{i+1}$ of states with a unique state in the unique preimage. By Lemma 7, we let $X \subseteq Q \cdot w_{i+1}$ to be a subset of size $2|Q \cdot w_{i+1}| - n = 2i + 2 - n$ of states $q \in Q \cdot w_{i+1}$ such that $|q \cdot w_{i+1}^{-1}| = 1$. We set $w_i = v_X w_{i+1}$, where v_X is the avoiding word from the assumption of the lemma for set X. We have $p \notin Q \cdot v_X$ for some $p \in X$.

State p is the only state mapped by the transformation of w_{i+1} to some state $q = p \cdot w_{i+1}$, i.e. there is no other state p' such that $p' \cdot w_{i+1} = q$. Hence we know that $q \notin Q \cdot w_i = Q \cdot v_X w_{i+1}$. Since $Q \cdot w_i \subseteq Q \cdot w_{i+1}$, $q \notin Q \cdot w_i$ but $q \in Q \cdot w_{i+1}$, we have $Q \cdot w_i \subsetneq Q \cdot w_{i+1}$. Therefore, we have rank

$$|Q \cdot w_i| \leq |Q \cdot w_{i+1}| - 1 \leq i + 1 - 1 = i,$$

and length

$$\begin{aligned} |w_i| &\leq k(n - |A|) + |w_{i+1}| \\ &\leq 2k(n-i-1) + k(r-(i+1))(2n-r-(i+1)-1) + |w| \\ &= k(r-i)(2n-r-i-1) + |w|. \end{aligned}$$

Finally, for $i = \lfloor n/2 \rfloor$ we obtain:

$$\begin{aligned} &|w| + k(r - \lfloor n/2 \rfloor)(2n - r - \lfloor n/2 \rfloor - 1) \\ &\leq |w| + k(r - (n-1)/2)(2n - r - (n-1)/2 - 1) \\ &= |w| + k(n^2 - (2n - 2r - 1)^2)/4. \end{aligned}$$ ◀

Note that Lemma 4 also provides an upper bound on the length of the shortest avoiding words, but it is larger than that the corresponding bound from Theorem 1, and so would not yield an improvement when used as in Lemma 8. Therefore, we use there an assumption about the length of the shortest avoiding words.

We observe that it is profitable to use Theorem 1 to find the starting word w, as long as $C(i+1,i)$ is smaller than $k(n-|A|)$. An approximate solution is to find the starting word w of rank at most $n-4k$. The following lemma utilizes this idea.

▶ **Lemma 9.** *Suppose that for some integer k, $1 \leq k \leq n/8$, for every $A \subset Q$ of size $1 \leq |A| \leq n-1$, there is a word $v_A \in \Sigma^{\leq k(n-|A|)}$ such that $A \not\subseteq Q \cdot v_A$. Then there is a word of rank at most $n/2$ and length at most*

$$k\frac{3n^2-64k^2+144k+13}{12}.$$

Proof. From Theorem 1, let w be a word of rank at most $n-4k$ and length at most

$$C(n,n-4k) = 4k(8k^2+6k+1)/3.$$

If w has rank $\geq \lfloor (n+1)/2 \rfloor$, then we apply Lemma 8 and obtain a word of rank at most $n/2$ and length at most

$$\begin{aligned}&\frac{4k(8k^2+6k+1)}{3} + \frac{k(n^2-(2n-2(n-4k)-1)^2)}{4}\\ =\ &\frac{k(3n^2-64k^2+144k+13)}{12}.\end{aligned}$$

Otherwise, w has rank $< n/2$, and because

$$k(n^2-(2n-2(n-4k)-1)^2)/4 = k(n^2-(8k-1)^2)/4$$

is positive for $1 \leq k \leq n/8$ (and $n \geq 8$), the upper bound is also valid. Thus, w has the desired length. ◀

We prove a parametrized upper bound on the reset threshold, depending on whether the assumption in Lemma 9 holds. When the assumption holds, the lemma provides an upper bound using avoiding words; otherwise, we have a quadratic word of a particular rank that yields an improvement.

▶ **Lemma 10.** *For every integer $1 \leq k \leq n/8$, there exists a reset word of length at most*

$$\max\left\{k\frac{3n^2-64k^2+144k+13}{12},\ k(n-1)+C(n-k,\lfloor n/2 \rfloor)\right\} + C(\lfloor n/2 \rfloor, 1).$$

Proof. We use Lemma 3 with the given k and subset $S = Q$.

Suppose that Case (1) from Lemma 3 holds for every $A \subset Q$ with $1 \leq |A| \leq n-1$. Then by Lemma 9 we obtain a word w of rank $\leq n/2$ and length $\leq k(3n^2-64k^2+144k+13)/12$.

Suppose that Case (2) from Lemma 3 holds for some $A \subset Q$ with $1 \leq |A| \leq n-1$. Then we have a word w of rank $\leq n-k$ and length $\leq k(n-1)$. By Theorem 1, we construct a word compressing $Q \cdot w$ to a subset of size $\leq n/2$. Then $k(n-1)+C(n-k,\lfloor n/2 \rfloor)$ is an upper bound for the length of the found word of rank $\leq n/2$.

Finally, we need to take the maximum from both cases, and add $C(\lfloor n/2 \rfloor, 1)$ to bound the length of a word compressing a subset of size $\lfloor n/2 \rfloor$ to a singleton. ◀

Now, by finding a suitable k, we state the new general upper bound on the reset threshold:

▶ **Theorem 11.**

$$\mathrm{rt}(\mathscr{A}) \leq (85059n^3+90024n^2+196504n-10648)/511104.$$

Proof. We use Lemma 10 with a suitable k that minimizes the maximum for large enough n.

First, we bound $C(n-k, \lfloor n/2 \rfloor)$ in the second argument in the maximum. If n is even then

$$\begin{aligned} C(n-k, \lfloor n/2 \rfloor) &= C(n-k, n/2) \\ &= \sum_{s=n/2+1}^{n-k} \frac{(n-s+2)(n-s+1)}{2} \\ &= \frac{n^3+6n^2+8n-8k^3-24k^2-16k}{48}. \end{aligned}$$

If n is odd then

$$\begin{aligned} C(n-k, \lfloor n/2 \rfloor) &= C(n-k, (n-1)/2) \\ &= \sum_{s=(n-1)/2+1}^{n-k} \frac{(n-s+2)(n-s+1)}{2} \\ &= \frac{n^3+9n^2+23n-8k^3-24k^2-16k+15}{48}, \end{aligned}$$

which is larger than the previous one.

Now we discuss our choice of k; any value of k gives a bound but we try to get it minimal. Assume that n is large enough. Note that for the largest possible value $k = n/8$ the first function in the maximum from Lemma 10 yields the coefficient of n^3 equal to $1/48$ (the same as by $C(n, \lfloor n/2 \rfloor)$), hence does not give an improvement. For a similar reason, we reject small values $k \in o(n)$. Within linear values k of n, the first function decreases and the second function increases with k. Since they are continuous, it is enough to consider the values of k such that both functions are equal. The approximate solution is $k \simeq 0.11375462n$. For simplicity of the calculations and the final formula, we use the approximation $k = \lfloor 5/44n \rfloor$. Note that any value of k within the valid range will lead to a correct bound, and we use $5/44$ since it is the best approximation by a rational number using integers with at most two digits.

We assume $n \geq 9$; for the smaller values of n the bound is a valid upper bound since it gives larger values than the bound from Theorem 1.

In the following calculations, we use the fact that $5/44n - 1 < \lfloor 5/44n \rfloor$ and $5/44n - 1$ is non-negative. By substitution, for the first function in the maximum we have

$$\begin{aligned} & k\frac{3n^2-64k^2+144k+13}{12} \\ &< (5/44n)\frac{3n^2-64(5/44n-1)^2+144(5/44n)+13}{12} \\ &= (5n(263n^2+3740n-6171))/63888, \end{aligned} \tag{1}$$

and for the second function we have

$$\begin{aligned} & k(n-1) + \frac{n^3+9n^2+23n-8k^3-24k^2-16k+15}{48} \\ &< (5/44n)(n-1) + \big(n^3+9n^2+23n-8(5/44n-1)^3 \\ &\quad -24(5/44n-1)^2-16(5/44n-1)+15\big)/48 \\ &= (10523n^3+153912n^2+196504n+159720)/511104. \end{aligned} \tag{2}$$

Note that (2) is larger than (1) for all n.

Now we have to bound $C(\lfloor n/2 \rfloor, 1)$. If n is even then

$$C(\lfloor n/2 \rfloor, 1) = C(n/2, 1) = (7n^3 - 6n^2 - 16)/48.$$

If n is odd then

$$C(\lfloor n/2 \rfloor, 1) = C((n-1)/2, 1) = (7n^3 - 9n^2 - 31n - 15)/48,$$

which is smaller than the previous one for $n \geq 2$.

Finally, we obtain

$$\frac{10523n^3 + 152262n^2 + 189244n + 191664}{511104} + \frac{7n^3 - 6n^2 - 16}{48}$$
$$= \frac{85059n^3 + 90024n^2 + 196504n - 10648}{511104}.$$ ◀

The theorem improves the old well known bound $(n^3 - n)/6 - 1$ by the factor 85059/85184, or by the coefficient 125/511104 of n^3. This is slightly better than the simpler formula $114n^3/685 + O(n^2)$.

The bound does not necessarily apply for the words obtained by a greedy compression algorithm for synchronization ([1, 11]), because the words in the proof of Lemma 8 are constructed by appending avoiding words at the beginning. However, we can show that there exists a polynomial algorithm finding words of lengths within the bound.

▶ **Proposition 12.** *A reset word of length within the bound from Theorem 11 can be computed in polynomial time.*

Proof. We use k from the proof of Theorem 11. We follow the construction from the proof of Lemma 8. By Theorem 6, we can compute a word from Lemma 2 for a subset A. If (1) holds every time, then we use the obtained word from Lemma 8. Otherwise, we use the word from Lemma 2 for which (2) holds. Finally, the words of lengths at most $C(j, i)$ are computed using a greedy compression algorithm ([1]). ◀

4 Further remarks

Although the improvement in terms of the cubic coefficient is small, it breaks longstanding persistence of the old bound from [21], and possibly opens the area for further progress.

Tiny improvements of the bound from Theorem 11 are possible with more effort yielding better calculations, for example by tuning the value of k in Theorem 11, better rounding, using better bounds at the beginning (note that one can find a shorter word than the word of rank k when Case (2) holds in Lemma 3 by combining with Theorem 1). These however do not add new ideas.

Avoiding a state

The first natural possibility for improving the bound is to show a better bound on the length of the shortest avoiding words. For strongly connected synchronizing automata, currently the best known lower bound is $2n - 3$ by Vojtěch Vorel[1] (binary series), whereas $2n - 2$ is conjectured to be a tight upper bound based on experiments [16].

▶ **Open Problem 1.** *Is $2n - 2$ the tight upper bound on the length of the shortest avoiding words?*

[1] personal communication, unpublished, 2016

Avoiding a subset

The technique from Lemma 8 can be applied only for compressing Q to a subset of size at most $n/2$, because at this point there can be no states with a unique state in the preimage. To bypass this obstacle, we can generalize the concept of avoiding to subsets, and say that a word w *avoids* a subset $D \subseteq Q$ if $D \cap (Q \cdot w) = \emptyset$. Having a good upper bound on the length of the shortest words avoiding D, we could continue using avoiding words for subsets smaller than $n/2$, since for a word s there are at least $|D| \cdot |Q \cdot s| - n$ states such that $1 \leq |q \cdot s^{-1}| \leq |D|$ (see Lemma 7).

▶ **Open Problem 2.** *Find a good upper bound (in terms of $|D|$ and n) on the length ℓ such that in every n-state automaton, for every subset $D \subset Q$ there is a word avoiding D of length at most ℓ, unless D is not avoidable.*

In fact, we can prove an upper bound in the spirit of Lemma 2, provided that we have avoiding words for smaller subsets than D.

▶ **Lemma 13.** *For $n \geq 2$, let $\mathscr{A}(Q, \Sigma, \delta)$ be an n-state strongly connected synchronizing automaton. Consider non-empty subsets $S, D \subseteq Q$ such that $|S| \geq 1$ and $|D| \geq 2$. Suppose that there is a state $p \in D$ such that for $D' = D \setminus \{p\}$ there exists a word $w_{D'} \in \Sigma^{\ell}$ that avoids D'. Then there exists a word $w \in \Sigma^{n-1+\ell}$ such that either:*
1. *$(S \cdot w) \cap D = \emptyset$, or*
2. *$|S \cdot w| < |S|$.*

Proof. Let $L_i = \operatorname{span}(\{[S][w] \mid w \in \Sigma^{\leq i}\})$. We consider the following sequence of linear subspaces:

$$L_0 \subseteq L_1 \subseteq L_2 \subseteq \dots,$$

and use the ascending chain condition as in the proof of Lemma 2. Since the automaton is synchronizing, there is a reset word u so $[S][u] = n[q]$ for some state q. Since the automaton is strongly connected, for every state p we have a word v such that $q \cdot v = p$, and so $[S][uv] = n[p]$. These vectors generate the whole space $\mathbb{R}^n$, and so the maximal dimension of the linear subspaces from the sequence is n; in particular, $\dim(L_{n-1}) = n$.

Let $P = p \cdot (w_{D'})^{-1}$. Suppose for a contradiction that for every word w of length $\leq n-1$, subset S is not compressed by w and $|(S \cdot w) \cap P| = 1$. Then $[S][w]$ contains exactly one 1 and $|P| - 1$ 0s at the positions corresponding to the states from P. Therefore, all vectors v generated by the vectors with this property satisfy:

$$(|S| - 1) \sum_{i \in P} v(i) = \sum_{i \in Q \setminus P} v(i).$$

This means that the dimension of L_{n-1} is at most $n-1$, since in $\mathbb{R}^n$ there are vectors that broke this equality. Hence, we have a contradiction.

Hence, there must be a word w that either compresses S or is such that $|(S \cdot w) \cap P| \neq 1$. In the latter case, if $(S \cdot w) \cap P = \emptyset$ then we obtain $(S \cdot ww_{D'}) \cap D = \emptyset$. If $(S \cdot w) \cap P \geq 2$ then $w_{D'}$ maps at least two states from $(S \cdot w) \cap P$ to p, thus $ww_{D'}$ compresses S. ◀

By an iterative application of the above lemma, we can obtain the upper bound $k(n-1+kn)$ on the length of a word that either avoids two states from the given subset or compresses the subset. This bound is too large to provide a further improvement (at least within the cubic coefficient) for the upper bound on the length of the shortest reset words. However, if the shortest words avoiding a single state are indeed of linear length, then we obtain a quadratic upper bound on the length of the shortest words avoiding two states.

References

1 D. S. Ananichev and V. V. Gusev. Approximation of Reset Thresholds with Greedy Algorithms. *Fundamenta Informaticae*, 145(3):221–227, 2016.

2 D. S. Ananichev and M. V. Volkov. Synchronizing generalized monotonic automata. *Theoretical Computer Science*, 330(1):3–13, 2005.

3 M.-P. Béal, M. V. Berlinkov, and D. Perrin. A quadratic upper bound on the size of a synchronizing word in one-cluster automata. *International Journal of Foundations of Computer Science*, 22(2):277–288, 2011.

4 M. Berlinkov and M. Szykuła. Algebraic synchronization criterion and computing reset words. *Information Sciences*, 369:718–730, 2016.

5 M. V. Berlinkov. Synchronizing Quasi-Eulerian and Quasi-one-cluster Automata. *International Journal of Foundations of Computer Science*, 24(6):729–745, 2013.

6 M. T. Biskup and W. Plandowski. Shortest synchronizing strings for Huffman codes. *Theoretical Computer Science*, 410(38-40):3925–3941, 2009.

7 J. Černý. Poznámka k homogénnym eksperimentom s konečnými automatami. *Matematicko-fyzikálny Časopis Slovenskej Akadémie Vied*, 14(3):208–216, 1964. In Slovak.

8 J. Černý, A. Pirická, and B. Rosenauerová. On directable automata. *Kybernetica*, 7:289–298, 1971.

9 H. Don. The Černý Conjecture and 1-Contracting Automata. *Electronic Journal of Combinatorics*, 23(3):P3.12, 2016.

10 L. Dubuc. Sur les automates circulaires et la conjecture de Černý. *Informatique théorique et applications*, 32:21–34, 1998. In French.

11 D. Eppstein. Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19:500–510, 1990.

12 P. Frankl. An extremal problem for two families of sets. *European Journal of Combinatorics*, 3:125–127, 1982.

13 F. Gonze, R. M Jungers, and A. N. Trahtman. A Note on a Recent Attempt to Improve the Pin-Frankl Bound. *Discrete Mathematics and Theoretical Computer Science*, 17(1):307–308, 2015.

14 M. Grech and A. Kisielewicz. The Černý conjecture for automata respecting intervals of a directed graph. *Discrete Mathematics and Theoretical Computer Science*, 15(3):61–72, 2013.

15 J. Kari. Synchronizing finite automata on Eulerian digraphs. *Theoretical Computer Science*, 295(1-3):223–232, 2003.

16 A. Kisielewicz, J. Kowalski, and M. Szykuła. Experiments with Synchronizing Automata. In *Implementation and Application of Automata*, volume 9705 of *LNCS*, pages 176–188. Springer, 2016.

17 A. A. Klyachko, I. K. Rystsov, and M. A. Spivak. An extremal combinatorial problem associated with the bound on the length of a synchronizing word in an automaton. *Cybernetics*, 23(2):165–171, 1987.

18 P. V. Martugin. A series of slowly synchronizing automata with a zero state over a small alphabet. *Information and Computation*, 206(9-10):1197–1203, 2008.

19 J.-E. Pin. Sur les mots synchronisants dans un automate fini. *Elektron. Informationsverarb. Kybernet.*, 14:293–303, 1978.

20 J.-E. Pin. Utilisation de l'algèbre linéaire en théorie des automates. In *Actes du 1er Colloque AFCET-SMF de Mathématiques Appliquées II*, AFCET, pages 85–92, 1978. In French.

21 J.-E. Pin. On two combinatorial problems arising from automata theory. In *Proceedings of the International Colloquium on Graph Theory and Combinatorics*, volume 75 of *North-Holland Mathematics Studies*, pages 535–548, 1983.

22 I. K. Rystsov. Reset words for commutative and solvable automata. *Theoretical Computer Science*, 172(1-2):273–279, 1997.

23 P. H. Starke. Eine Bemerkung über homogene Experimente. *Elektronishe Informationverarbeitung und Kybernetic*, 2:257–259, 1966. In German.

24 B. Steinberg. The averaging trick and the Černý conjecture. *International Journal of Foundations of Computer Science*, 22(7):1697–1706, 2011.

25 B. Steinberg. The Černý conjecture for one-cluster automata with prime length cycle. *Theoretical Computer Science*, 412(39):5487–5491, 2011.

26 A. N. Trahtman. The Černý conjecture for aperiodic automata. *Discrete Mathematics and Theoretical Computer Science*, 9(2):3–10, 2007.

27 A. N. Trahtman. Modifying the upper bound on the length of minimal synchronizing word. In *Fundamentals of Computation Theory*, volume 6914 of *LNCS*, pages 173–180. Springer, 2011.

28 M. V. Volkov. Synchronizing automata and the Černý conjecture. In *Language and Automata Theory and Applications*, volume 5196 of *LNCS*, pages 11–27. Springer, 2008.

29 M. V. Volkov. Synchronizing automata preserving a chain of partial orders. *Theoretical Computer Science*, 410(37):3513–3519, 2009.

Power of Uninitialized Qubits in Shallow Quantum Circuits

Yasuhiro Takahashi
NTT Communication Science Laboratories, NTT Corporation, Japan
takahashi.yasuhiro@lab.ntt.co.jp

Seiichiro Tani
NTT Communication Science Laboratories, NTT Corporation, Japan
tani.seiichiro@lab.ntt.co.jp

Abstract

We study the computational power of shallow quantum circuits with $O(\log n)$ initialized and $n^{O(1)}$ uninitialized ancillary qubits, where n is the input length and the initial state of the uninitialized ancillary qubits is arbitrary. First, we show that such a circuit can compute any symmetric function on n bits that is classically computable in polynomial time. Then, we regard such a circuit as an oracle and show that a polynomial-time classical algorithm with the oracle can estimate the elements of any unitary matrix corresponding to a constant-depth quantum circuit on n qubits. Since it seems unlikely that these tasks can be done with only $O(\log n)$ initialized ancillary qubits, our results give evidences that adding uninitialized ancillary qubits increases the computational power of shallow quantum circuits with only $O(\log n)$ initialized ancillary qubits. Lastly, to understand the limitations of uninitialized ancillary qubits, we focus on near-logarithmic-depth quantum circuits with them and show the impossibility of computing the parity function on n bits.

2012 ACM Subject Classification Theory of computation → Quantum computation theory

Keywords and phrases Quantum Circuit Complexity, Shallow Quantum Circuit, Uninitialized Qubit

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.57

Related Version A full version of the paper is available at `https://arxiv.org/abs/1608.07020`.

1 Introduction

1.1 Background and Main Results

Much attention has been paid to the computational power of shallow (i.e., polylogarithmic-depth) quantum circuits [6, 16, 10, 9, 11, 8, 3, 22, 20, 4]. A major purpose of this line of research is to understand the differences between shallow quantum and classical circuits. In addition, it is strongly motivated by one of the most difficult problems concerning quantum circuit implementation: in current and near-future technologies, it would be very difficult to keep quantum coherence for a period of time long enough to apply many gates.

In discussing the computational power of shallow quantum circuits, polynomially many ancillary qubits initialized to, say, $|0\rangle$ are assumed to be available. The initialized ancillary qubits are particularly important for quantum circuits since many quantum operations require ancillary qubits to preserve their unitary property and store intermediate results. Another implementation problem arises here: it is difficult to prepare a large number of qubits that are simultaneously initialized to a certain state. Indeed, this problem has often been addressed

in the literature [7, 14]. However, most papers concerning the problem assume a sufficiently long coherence time. In this paper, we address these two problems simultaneously.

A straightforward quantum computation model reflecting a short coherence time and a limited number of initialized ancillary qubits would be shallow quantum circuits with $O(\log n)$ initialized ancillary qubits, where n is the input length. However, their computational power seems quite low since each step of them can utilize only a small number of intermediate results. In fact, it is not even known whether such a circuit can compute the OR function on n bits, and it seems unlikely that it can. Therefore, it is highly desirable to find additional ancillary qubits satisfying the following conditions: they should be easier to prepare than initialized ancillary qubits and increase the computational power of shallow quantum circuits with only $O(\log n)$ initialized ancillary qubits. An interesting direction is to study qubits in the completely mixed state [13], but it would be better not to assume any particular initial state.

We consider polynomially many uninitialized qubits as additional ancillary qubits. More concretely, we study shallow quantum circuits with $O(\log n)$ initialized and $n^{O(1)}$ uninitialized ancillary qubits, where we assume that no intermediate measurements are allowed. The initial state of the uninitialized ancillary qubits is arbitrary and thus they are easier to prepare than initialized ancillary qubits, i.e., they satisfy the above first condition on additional ancillary qubits. But do they satisfy the second condition? Specifically, are shallow quantum circuits with $O(\log n)$ initialized and $n^{O(1)}$ uninitialized ancillary qubits more powerful than those without uninitialized ancillary qubits? Although uninitialized ancillary qubits are known to be useful for constructing a few efficient quantum circuits [1, 19], a complexity-theoretic analysis of quantum circuits with such ancillary qubits has not yet been done.

First, to give evidence of an affirmative answer to the question, we consider symmetric functions, which are Boolean functions whose output depends only on the number of ones in the input bits [12]. Let $\mathcal{S}_n$ be the class of symmetric functions on n bits that are classically computable in polynomial time. For example, $\mathcal{S}_n$ includes the OR function, for which it is not known whether there exists a shallow quantum circuit (consisting of one-qubit gates and CNOT gates) with only $O(\log n)$ initialized ancillary qubits, and it seems unlikely that it does. However, any function in $\mathcal{S}_n$ can be computed by adding uninitialized ancillary qubits:

▶ **Theorem 1.** *Any $f_n \in \mathcal{S}_n$ can be computed by an $O((\log n)^2)$-depth quantum circuit with n input qubits, one output qubit, and $O(\log n)$ initialized and $O(n(\log n)^2)$ uninitialized ancillary qubits such that it consists of the gates in the gate set $\mathcal{G}$, where $\mathcal{G}$ consists of a Hadamard gate, a phase-shift gate with angle $2\pi c/2^t$ for any integers $t \geq 1$ and c, and a CNOT gate.*

Theorem 1 gives evidence that shallow quantum circuits with $O(\log n)$ initialized and $n^{O(1)}$ uninitialized ancillary qubits are more powerful than those without uninitialized ancillary qubits in terms of computing symmetric functions. The proof of Theorem 1 immediately implies that the depth of the circuit can be decreased to $O(\log n)$ when the circuit is allowed to further include unbounded fan-out gates and unbounded Toffoli gates.

Then, to give further evidence of the computational advantage of using uninitialized ancillary qubits, we consider a classical algorithm with an oracle that can perform a shallow quantum circuit with them. When the oracle receives a bit string w, it performs the circuit with input qubits initialized to $|w\rangle$ and sends back the classical outcome of the measurement on the output qubit. Let $p(n)$ be a polynomial and C_n be a constant-depth quantum circuit on n qubits consisting of the gates in $\mathcal{G}$. The problem, denoted by $\mathrm{MAT}(p(n), C_n)$, is to compute a real number α_x such that $|\alpha_x - |\langle 0^n|C_n|x\rangle|^2| \leq 1/p(n)$ for any input $x \in \{0,1\}^n$, where C_n also denotes its matrix representation. It is not known whether the problem has a polynomial-time classical algorithm, and it seems unlikely that it does [17], even when we use

an oracle that can perform a shallow quantum circuit with only $O(\log n)$ initialized ancillary qubits. However, the problem can be solved by adding uninitialized ancillary qubits:

▶ **Theorem 2.** *For any polynomial $p(n)$ and a constant-depth quantum circuit C_n on n qubits consisting of the gates in $\mathcal{G}$, $\mathrm{MAT}(p(n), C_n)$ can be solved with probability exponentially (in n) close to 1 by a polynomial-time probabilistic classical algorithm with an oracle that can perform an $O(\log n)$-depth quantum circuit with $2n$ input qubits, one output qubit, and (no initialized and) n uninitialized ancillary qubits such that it consists of the gates in $\mathcal{G}$.*

As with Theorem 1, Theorem 2 gives evidence that shallow quantum circuits with $O(\log n)$ initialized and $n^{O(1)}$ uninitialized ancillary qubits are more powerful than those without uninitialized ancillary qubits. More concretely, by the proof of Theorem 2, this is evidence that there exists a probability distribution on $\{0,1\}$ that can be generated with uninitialized ancillary qubits but cannot without them. This is because, otherwise, $\mathrm{MAT}(p(n), C_n)$ would be solved by using an oracle with only $O(\log n)$ initialized ancillary qubits. We give a brief comment on the number of input qubits in the circuit performed by the oracle. If the number is large, a classical algorithm can send 0^k for large k (besides another bit string) to the oracle and the circuit can use a part of the input qubits as a large number of initialized ancillary qubits. To avoid this, we restrict the number of input qubits to $2n$.

Lastly, to understand the limitations of uninitialized ancillary qubits, for an arbitrary constant $0 \leq \delta < 1$, we focus on $O((\log n)^\delta)$-depth quantum circuits with them and consider the computability of the parity function on n bits. Since the depth is $o(\log n)$, it is easy to show that the parity function cannot be computed by any such circuit consisting of the gates in $\mathcal{G}$. This is also the case even when the circuit includes additional gates on a non-constant number of qubits:

▶ **Theorem 3.** *Let $0 \leq \delta < 1$ be an arbitrary constant. Then, the parity function on n bits cannot be computed by any $O((\log n)^\delta)$-depth quantum circuit with n input qubits, one output qubit, and $O(\log n)$ initialized and $n^{O(1)}$ uninitialized ancillary qubits such that it consists of the gates in $\mathcal{G}$, unbounded fan-out gates on $(\log n)^{O(1)}$ qubits, and unbounded Toffoli gates.*

Theorem 3 means that $O((\log n)^\delta)$-depth quantum circuits with $O(\log n)$ initialized and $n^{O(1)}$ uninitialized ancillary qubits are *not* more powerful than those without uninitialized ancillary qubits in terms of computing the parity function, even when they include the two types of gates on a non-constant number of qubits. Moreover, Theorem 3 implies that the circuit in Theorem 1 is optimal in the following sense. As described in the paragraph following Theorem 1, the depth of the circuit becomes $O(\log n)$ when the circuit uses the gates in $\mathcal{G}$, unbounded fan-out gates, and unbounded Toffoli gates. As described in Section 1.3, the circuit is based on the computation of the number of ones in the input bits and thus can be regarded as a parity circuit. Thus, the circuit cannot be significantly improved simultaneously in terms of both the depth and the number of qubits on which unbounded fan-out gates act. This is because, otherwise, we would obtain a parity circuit that contradicts Theorem 3.

We comment on the states of uninitialized ancillary qubits in the above theorems. In the proofs of Theorems 1 and 2, we assume that the state of uninitialized ancillary qubits is an arbitrary computational basis (pure) state. These proofs can be simply extended for an arbitrary pure/mixed state by the linearity of quantum operations and the fact that a mixed state is a probabilistic mixture of pure states. Thus, Theorems 1 and 2 hold for an arbitrary pure/mixed state. We show Theorem 3 under the same assumption. Thus, Theorem 3 means that there does not exist an $O((\log n)^\delta)$-depth quantum circuit (with the property described in the theorem) that computes the parity function on n bits regardless of the initial state

of uninitialized ancillary qubits, where we assume that the initial state is restricted to an arbitrary computational basis (pure) state. In this statement, we can remove the restriction, i.e., we can assume that the initial state is an arbitrary pure/mixed state. This is because the resulting statement is weaker than the original one. In this sense, Theorem 3 holds for an arbitrary pure/mixed state.

1.2 Imposing the Quantum Catalytic Requirement

Buhrman et al. [5] defined a *classical* computation with a logarithmic-size clean space and a polynomial-size additional space, which they call a catalytic log-space computation. The initial state of the additional space is arbitrary, and they impose the catalytic requirement that its state has to be returned to the initial one at the end of the computation. They showed a surprising result: it appears that such a computation is more powerful than that without the additional space. The additional space seems like a catalyst in a chemical reaction.

The corresponding catalytic requirement in our quantum setting is that the state of uninitialized ancillary qubits has to be returned to the initial one at the end of computation. Since the circuit in Theorem 1 has no error, by the standard technique of uncomputation, it is easy to transform the circuit into the one that meets the quantum catalytic requirement without increasing the original asymptotic complexity. Thus, Theorem 1 means that uninitialized ancillary qubits seem like a catalyst as in the classical setting [5]. When shallow quantum circuits have an error, it is not easy to transform them into the ones that meet the quantum catalytic requirement and the analysis of such circuits is left for future work.

From a practical point of view, it is even better to decrease the number of uninitialized ancillary qubits we need to specially prepare in addition to decreasing the number of initialized ones. The quantum catalytic requirement allows us to do this in some cases. An example is when we use a shallow quantum circuit with uninitialized ancillary qubits in a quantum circuit for Shor's factoring algorithm [19]. The factoring circuit uses two registers and, during some operation, all qubits in one register are idle. Thus, when we use a shallow quantum circuit for the operation that meets the above requirement, we can regard the idle qubits as uninitialized ancillary qubits since the circuit returns their state to the initial one. The use of the circuit in this way requires that the computation has to be done with only qubits, which matches our quantum computation model. From a complexity-theoretic standpoint, it is also interesting to study a quantum computation model with an additional classical space [23].

1.3 Overview of Techniques

We construct two quantum circuits to obtain the circuit for $f_n \in \mathcal{S}_n$ in Theorem 1. The first one is an $O((\log n)^2)$-depth OR reduction circuit with $O(n(\log n)^2)$ uninitialized ancillary qubits, which reduces the computation of the OR function on n bits to that on $m = O(\log n)$ bits. Its first part is a modification of the original OR reduction circuit [11] and yields a state whose phase depends on the uninitialized ancillary qubits but has a convenient form to eliminate the dependency. We apply similar circuits repeatedly to add an appropriate phase to that of the state, which eliminates any dependency on the uninitialized ancillary qubits. The second circuit is an $O(m^2)$-depth one for g_m with $O(m2^m)$ uninitialized ancillary qubits. Here, g_m is a Boolean function on m bits satisfying that $g_m(s) = f_n(x)$ for any $x \in \{0,1\}^n$, where $s \in \{0,1\}^m$ is the binary representation of the number of ones in x. The circuit is based on the Fourier expansion of g_m [12] and the above method for eliminating any dependency on the uninitialized ancillary qubits. For any input $x \in \{0,1\}^n$, we first compute s using the OR reduction circuit and then compute $g_m(s) = f_n(x)$ using the circuit for g_m.

The algorithm in Theorem 2 is based on a polynomial-time probabilistic classical algorithm for $\mathrm{MAT}(p(n), C_n)$ with an oracle [17], where the oracle can perform a commuting quantum circuit for the Hadamard test [15]. Although initialized ancillary qubits can be used to parallelize the Hadamard test [22], it has not been known whether uninitialized ancillary qubits are useful for this purpose. We show that they can be used like initialized ancillary qubits in parallelizing the Hadamard test. We replace the commuting quantum circuit with a new circuit with our parallelizing techniques using uninitialized ancillary qubits in the algorithm for $\mathrm{MAT}(p(n), C_n)$, which yields the desired algorithm.

We show Theorem 3 by extending the proof of Bera [3]. Our proof is different from the previous one in that it deals with ancillary qubits and unbounded fan-out gates. The key to Theorem 3 is to show that, for any quantum circuit C_n with $O(\log n)$ initialized and $n^{O(1)}$ uninitialized ancillary qubits such that it may include unbounded Toffoli gates, there exists an initial state of the uninitialized ancillary qubits such that C_n with the initial state is well approximated by $\tilde{C}_n$ with the same initial state. Here, $\tilde{C}_n$ is the circuit obtained from C_n by removing unbounded Toffoli gates on a large number of qubits. Thus, if C_n is a small-depth quantum circuit for the parity function, then $\tilde{C}_n$ computes the same function with high probability. This is impossible since $\tilde{C}_n$ does not have any gate on a large number of qubits and thus its output does not depend on all input qubits.

2 Preliminaries

A quantum circuit consists of elementary gates, each of which is in the gate set $\mathcal{G}$, where $\mathcal{G}$ consists of a Hadamard gate H, a phase-shift gate $Z(\theta)$ with angle θ, and a CNOT gate. Here, $H = |+\rangle\langle 0| + |-\rangle\langle 1|$ and $Z(\theta) = |0\rangle\langle 0| + e^{i\theta}|1\rangle\langle 1|$, where $|\pm\rangle = (|0\rangle \pm |1\rangle)/\sqrt{2}$ and $\theta = 2\pi c/2^t$ for any integers $t \geq 1$ and c. We write $Z(\pi)$ and $HZ(\pi)H$ as Z and X, respectively. In some cases, we use a fan-out gate and a Toffoli gate as elementary gates. Let $k \geq 1$ be an integer. A fan-out gate on $k+1$ qubits implements the operation defined as $|y\rangle \bigotimes_{j=1}^{k} |x_j\rangle \mapsto |y\rangle \bigotimes_{j=1}^{k} |x_j \oplus y\rangle$ for any $y, x_j \in \{0,1\}$, where $\oplus$ denotes addition modulo 2. The first input qubit is called the control qubit. A k-controlled Toffoli gate implements the operation on $k+1$ qubits defined as $\left(\bigotimes_{j=1}^{k} |x_j\rangle\right)|y\rangle \mapsto \left(\bigotimes_{j=1}^{k} |x_j\rangle\right)|y \oplus \bigwedge_{j=1}^{k} x_j\rangle$, where $\bigwedge$ denotes the logical AND. The first k input qubits are called the control qubits and the last input qubit is called the target qubit. These gates with $k = 1$ are CNOT gates. When it is permitted to apply a fan-out gate and a Toffoli gate on a non-constant number of qubits, they are called an unbounded fan-out gate and an unbounded Toffoli gate, respectively.

To simplify the descriptions of quantum circuits, we use a k-controlled $Z(\theta)$ gate for any θ described above, which will be decomposed into elementary gates. The gate implements the operation on $k+1$ qubits defined as $\bigotimes_{j=1}^{k+1} |x_j\rangle \mapsto e^{i\theta \bigwedge_{j=1}^{k+1} x_j} \bigotimes_{j=1}^{k+1} |x_j\rangle$ for any $x_j \in \{0,1\}$. We can choose an arbitrary qubit as the target qubit and the other qubits are called the control qubits. The inverse of the gate is the k-controlled $Z(-\theta)$ gate. When it is permitted to apply the gate on a non-constant number of qubits, it is called an unbounded $Z(\theta)$ gate.

The complexity measures of a quantum circuit are its size and depth. The size of a quantum circuit is the total size of all elementary gates in the circuit, where the size of an elementary gate is the number of qubits on which the gate acts. To define the depth, we regard the circuit as a set of layers $1, \ldots, d$ consisting of elementary gates, where gates in the same layer act on pairwise disjoint sets of qubits and any gate in layer j is applied before any gate in layer $j+1$. The depth of the circuit is the smallest possible value of d [9].

We deal with a uniform family of polynomial-size quantum circuits $\{C_n\}_{n\geq 1}$, where no intermediate measurements are allowed. The uniformity means that the function $1^n \mapsto \overline{C_n}$ is classically computable in polynomial time, where $\overline{C_n}$ is the classical description of C_n. Each

C_n has n input qubits and can have one output qubit and $n^{O(1)}$ ancillary qubits that are divided into two groups: $p = O(\log n)$ qubits and the remaining q qubits. We assume that, for any $x \in \{0,1\}^n$ and $y \in \{0,1\}$, we can initialize the input qubits and output qubit to $|x\rangle$ and $|y\rangle$, respectively. We can also initialize the p ancillary qubits to $|0\rangle$, which we call initialized ancillary qubits, but we cannot initialize the q ancillary qubits and do not know their initial state. They are called uninitialized ancillary qubits. When C_n has the output qubit, a measurement in the Z basis is performed on it at the end of the computation. The classical outcome of the measurement, which is 0 or 1, is called the output of C_n. A symbol denoting a quantum circuit also denotes its matrix representation in the computational basis.

A Boolean function f_n on n bits is a mapping $f_n : \{0,1\}^n \to \{0,1\}$. We define its computability by a quantum circuit with uninitialized ancillary qubits as follows:

▶ **Definition 4.** Let f_n be a Boolean function on n bits and C_n be a quantum circuit with n input qubits, one output qubit, and p initialized and q uninitialized ancillary qubits. The circuit C_n computes f_n if, for any $x \in \{0,1\}^n$ and $y \in \{0,1\}$, when the input qubits and output qubit are initialized to $|x\rangle$ and $|y\rangle$, respectively, the output of C_n is $y \oplus f_n(x)$ with probability 1, regardless of the initial state of the q uninitialized ancillary qubits.

A Boolean function is called symmetric if its output depends only on the number of ones in the input bits [12]. Let $\mathcal{S}_n$ be the class of symmetric functions on n bits that are classically computable in polynomial time. For example, $\mathcal{S}_n$ includes the parity function PA_n and the OR function OR_n. Here, for any $x = x_1 \cdots x_n \in \{0,1\}^n$, $\mathrm{PA}_n(x) = 1$ if $|x|$ is odd and 0 otherwise, where $|x| = \sum_{j=1}^{n} x_j$. Moreover, $\mathrm{OR}_n(x) = 1$ if $|x| \geq 1$ and 0 otherwise.

We define the function associated with $f_n \in \mathcal{S}_n$ as follows:

▶ **Definition 5.** Let $f_n \in \mathcal{S}_n$. The function associated with f_n is the Boolean function g_m on $m = \lceil \log(n+1) \rceil$ bits defined as follows: For any $s = s_1 \cdots s_m \in \{0,1\}^m$, $g_m(s) = f_n(1^l 0^{n-l})$ if $l \leq n$ and 0 otherwise, where $l = \sum_{k=1}^{m} s_k 2^{k-1}$.

The function g_m is classically computable in time $n^{O(1)}$ and, for any $x \in \{0,1\}^n$, if $s = s_1 \cdots s_m$ is the binary representation of $|x|$, i.e., $|x| = \sum_{k=1}^{m} s_k 2^{k-1}$, then $g_m(s) = f_n(x)$.

We explain the idea of the original OR reduction quantum circuit [11]. The circuit has n input qubits and $O(n \log n)$ initialized ancillary qubits. Let $|x\rangle$ be an input state for any $x \in \{0,1\}^n$. The circuit transforms the state of m initialized ancillary qubits into the state $\bigotimes_{k=1}^{m} |\varphi_k\rangle$, where $m = \lceil \log(n+1) \rceil$ and $|\varphi_k\rangle = (|+\rangle + e^{\frac{2\pi i}{2^k}|x|}|-\rangle)/\sqrt{2}$ for any $1 \leq k \leq m$. If $|x| = 0$, then $|\varphi_k\rangle = |0\rangle$ for any $1 \leq k \leq m$ and thus the output state is $|0^m\rangle$. If $|x| \geq 1$, then $|\varphi_k\rangle = |1\rangle$ for some $1 \leq k \leq m$ and thus the output state is orthogonal to $|0^m\rangle$. This means that the circuit reduces the computation of OR_n to that of OR_m. The output state can be used to compute the binary representation $s_1 \cdots s_m$ of $|x|$. In fact, it is easy to show that the state $\bigotimes_{k=1}^{m} |s_k\rangle$ can be obtained by applying $\mathrm{QFT}_{2^m}^{\dagger}$ to the state $\bigotimes_{k=1}^{m} H|\varphi_k\rangle$, where $\mathrm{QFT}_{2^m}^{\dagger}$ is the inverse of the quantum Fourier transform modulo 2^m.

3 Shallow Quantum Circuits for Symmetric Functions

Let $f_n \in \mathcal{S}_n$. We compute f_n on input $x \in \{0,1\}^n$ using the following algorithm:

1. Compute the binary representation $s \in \{0,1\}^m$ of $|x|$, where $m = \lceil \log(n+1) \rceil$.
2. Compute $g_m(s) = f_n(x)$, where g_m is the function associated with f_n.

To implement Step 1, we construct an OR reduction circuit Q_n with uninitialized ancillary qubits. As described above, we can obtain s using Q_n (with a layer of H gates) and the standard $O(m)$-depth quantum circuit for $\mathrm{QFT}_{2^m}^{\dagger}$ with no ancillary qubits [18]. To implement Step 2, we construct a quantum circuit R_m for g_m with uninitialized ancillary qubits.

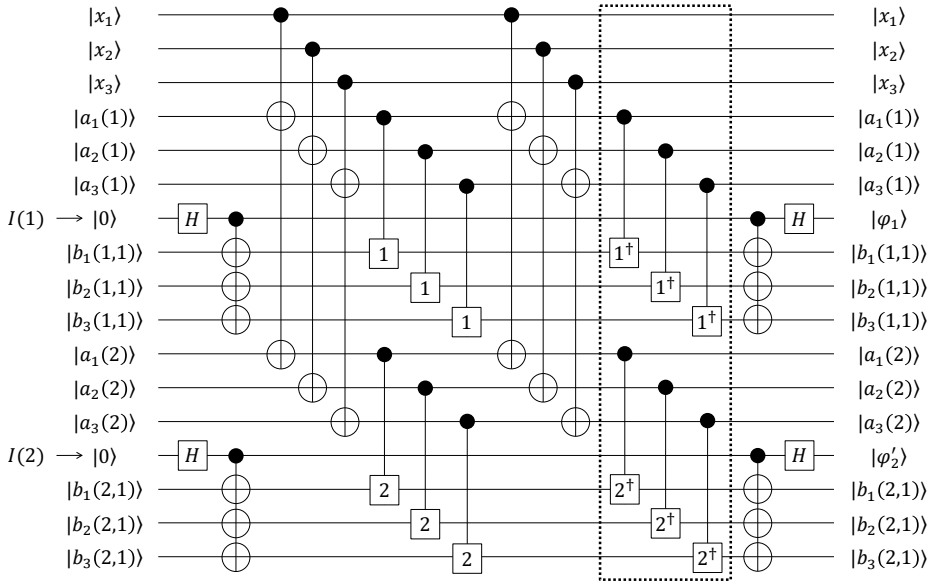


Figure 1 The first stage of our OR reduction circuit with input $x = x_1x_2x_3 \in \{0,1\}^3$. The gate next to the H gate is a fan-out gate on four qubits, where the top qubit is the control qubit. For any integer $t \geq 1$, the gates t and $t^\dagger$ represent a $Z(2\pi/2^t)$ gate and its inverse, i.e., a $Z(-2\pi/2^t)$ gate, respectively. The dashed box represents the gates added to the original OR reduction circuit.

3.1 OR Reduction Circuit with Uninitialized Ancillary Qubits

The circuit Q_n is an $O((\log n)^2)$-depth OR reduction circuit with n input qubits and $O(\log n)$ initialized and $O(n(\log n)^2)$ uninitialized ancillary qubits. To explain our idea for constructing Q_n, we consider the case where $n = 3$ (and thus $m = 2$). The first stage of Q_n is depicted in Fig. 1, where the initial state of the uninitialized ancillary qubits is represented by the (unknown) values $a_j(k), b_j(k,l) \in \{0,1\}$. This circuit is obtained by adding the gates in the dashed box to the original OR reduction circuit. We want to transform the initial states of the initialized ancillary qubits $I(1)$ and $I(2)$ into the states $|\varphi_1\rangle$ and $|\varphi_2\rangle$, respectively. If we do not apply the added gates, the output state of $I(k)$ is $(|+\rangle + e^{\frac{2\pi i}{2^k}\alpha(k,1)}|-\rangle)/\sqrt{2}$, where $\alpha(k,1) = \sum_{j=1}^{3}(-1)^{b_j(k,1)}(x_j \oplus a_j(k))$ and $k = 1, 2$. The phase of this state depends on the initial state of the uninitialized ancillary qubits and we eliminate the dependency.

The point is that the added gates allow us to obtain an output state of $I(k)$ whose phase has a convenient form to eliminate the dependency. More concretely, by applying them, the output state of $I(k)$ is $(|+\rangle + e^{\frac{2\pi i}{2^k}\gamma(k,1)}|-\rangle)/\sqrt{2}$, where $\gamma(k,1) = |x| - 2\sum_{j=1}^{3} x_j(a_j(k) \oplus b_j(k,1))$. Since $e^{\frac{2\pi i}{2}\gamma(1,1)} = e^{\frac{2\pi i}{2}|x|}$, the output state of $I(1)$ is equal to $|\varphi_1\rangle$ as desired. The dependency is eliminated since the terms in $\gamma(1,1)$ other than $|x|$ yield only an angle of a multiple of 2π.

Unfortunately, the output state of $I(2)$, which is represented as $|\varphi_2'\rangle$ in Fig. 1, is not equal to $|\varphi_2\rangle$ in general since the phase $e^{\frac{2\pi i}{2^2}\gamma(2,1)}$ depends on the initial states of the uninitialized ancillary qubits, where $\gamma(2,1) = |x| - 2\sum_{j=1}^{3} x_j(a_j(2) \oplus b_j(2,1))$. To eliminate the dependency, we consider the second stage where we add an angle $\frac{2\pi}{2^2}\delta(2,2)$ to the original angle $\frac{2\pi}{2^2}\gamma(2,1)$ using three new uninitialized ancillary qubits (not depicted in Fig. 1). Here, their initial state is $|b_1(2,2)\rangle|b_2(2,2)\rangle|b_3(2,2)\rangle$ for any (unknown) $b_j(2,2) \in \{0,1\}$ and $\delta(2,2) = |x| - \gamma(2,1) - 2^2\sum_{j=1}^{3} x_j(a_j(2) \oplus b_j(2,1))(a_j(2) \oplus b_j(2,2))$. The value $\delta(2,2)$ has a form similar to $\gamma(2,1)$ and thus we can implement the second stage using a quantum circuit similar to the one in Fig. 1. Since $e^{\frac{2\pi i}{2^2}(\gamma(2,1)+\delta(2,2))} = e^{\frac{2\pi i}{2^2}|x|}$, we obtain $|\varphi_2\rangle$ as desired.

We generalize the idea. Let $x = x_1 \cdots x_n \in \{0,1\}^n$ be an input. We prepare n input qubits $X_1, \ldots, X_n$ and m initialized ancillary qubits $I(1), \ldots, I(m)$, where X_j is initialized to $|x_j\rangle$. We also prepare $nm(m+3)/2$ uninitialized ancillary qubits, which are divided into two groups, A and B. Group A consists of mn qubits, which are divided into m groups $A(1), \ldots, A(m)$. Each $A(k)$ consists of n qubits $A_1(k), \ldots, A_n(k)$, where the initial state of $A_j(k)$ is $|a_j(k)\rangle$ for any (unknown) $a_j(k) \in \{0,1\}$. Group B consists of $nm(m+1)/2$ qubits, which are divided into m groups $B(1), \ldots, B(m)$. Each $B(k)$ consists of kn qubits, which are divided into k groups $B(k,1), \ldots, B(k,k)$. Each $B(k,l)$ consists of n qubits $B_1(k,l), \ldots, B_n(k,l)$, where the initial state of $B_j(k,l)$ is $|b_j(k,l)\rangle$ for any (unknown) $b_j(k,l) \in \{0,1\}$. The circuit Q_n consists of m stages. For any $1 \le s \le m$, Stage s is defined as follows:

1. Apply a H gate to $I(k)$ for every $s \le k \le m$ in parallel.
2. Apply a fan-out gate on $n+1$ qubits to $B_1(k,s), \ldots, B_n(k,s)$, and $I(k)$ for every $s \le k \le m$ in parallel, where $I(k)$ is the control qubit.
3. If $s \ge 2$, then apply a fan-out gate on s qubits to $B_j(k,1), \ldots, B_j(k,s-1)$, and $A_j(k)$ for every $s \le k \le m$ and $1 \le j \le n$ in parallel, where $A_j(k)$ is the control qubit.
4. Apply a fan-out gate on $m-s+2$ qubits to $A_j(s), A_j(s+1), \ldots, A_j(m)$, and X_j for every $1 \le j \le n$ in parallel, where X_j is the control qubit.
5. Apply an s-controlled $Z(2\pi/2^{k-s+1})$ gate to $B_j(k,s)$ and the following qubits for every $s \le k \le m$ and $1 \le j \le n$ in parallel: $A_j(k)$ if $s = 1$ and $B_j(k,1), \ldots, B_j(k,s-1)$, and $A_j(k)$ otherwise.
6. Apply the gates in Step 4.
7. Apply the inverse of the gates in Step 5.
8. Apply the gates in Step 3, Step 2, and Step 1 (in this order).

The circuit Q_n outputs the desired state and has the desired complexity as follows. The proofs can be found in the full version of the paper [21].

▶ **Lemma 6.** *Let $x = x_1 \cdots x_n \in \{0,1\}^n$ be an input. For any $1 \le k \le m$ and $1 \le s \le k$, the state of $I(k)$ after Stage s is the state $(|+\rangle + e^{\frac{2\pi i}{2^k}\gamma(k,s)}|-\rangle)/\sqrt{2}$, where $\gamma(k,s) = |x| - 2^s \sum_{j=1}^{n} x_j \bigwedge_{l=1}^{s}(a_j(k) \oplus b_j(k,l))$. Moreover, the state of any qubit other than the initialized ancillary qubits is the same as its initial one. The state of $I(k)$ after Stage k is the state $|\varphi_k\rangle$.*

▶ **Lemma 7.** *The circuit Q_n uses $O(\log n)$ initialized and $O(n(\log n)^2)$ uninitialized ancillary qubits, and its depth is $O((\log n)^2)$, when the elementary gate set is $\mathcal{G}$.*

3.2 Circuit for the Function Associated with a Symmetric Function

We construct an $O(m^2)$-depth quantum circuit R_m for g_m with $m = \lceil \log(n+1) \rceil$ input qubits, one output qubit, and $O(m2^m)$ uninitialized ancillary qubits, where g_m is the function associated with $f_n \in \mathcal{S}_n$. The circuit uses (a slight modification of) the Fourier expansion of g_m [12]: For any $s = s_1 \cdots s_m \in \{0,1\}^m$, $g_m(s) = g_m(0^m) + \frac{2}{2^m}\sum_t c_t \bigoplus_{k=1}^{m} t_k s_k$, where $c_t = \sum_u g_m(u)(2\bigoplus_{k=1}^{m} u_k t_k - 1)$, $t = t_1 \cdots t_m$ ranges over $\{0,1\}^m \setminus \{0^m\}$, and $u = u_1 \cdots u_m$ ranges over $\{0,1\}^m$. Since $m = O(\log n)$ and g_m is classically computable in time $n^{O(1)}$, the number of c_t's with $t \in \{0,1\}^m \setminus \{0^m\}$ is $n^{O(1)}$ and the function $t \mapsto c_t$ is also classically computable in time $n^{O(1)}$. This implies the uniformity of our circuit family for f_n.

The circuit R_m with input $s = s_1 \cdots s_m \in \{0,1\}^m$ is based on the following algorithm:

1. Compute the parity value $\bigoplus_{k=1}^{m} t_k s_k$ for every $t \in \{0,1\}^m \setminus \{0^m\}$ in parallel.
2. Prepare $(|+\rangle + e^{\pi i g_m(s)}|-\rangle)/\sqrt{2} = |g_m(s)\rangle$ using the above representation of g_m.

Since we do not have any initialized ancillary qubit, in Step 1, we can only have the parity values on uninitialized ancillary qubits, i.e., $a_t \oplus \bigoplus_{k=1}^{m} t_k s_k$ for every $t \in \{0,1\}^m \setminus \{0^m\}$, where the initial state of the uninitialized ancillary qubits is represented by the (unknown) values $a_t \in \{0,1\}$. Thus, in Step 2, we have to use such values to prepare $(|+\rangle + e^{\pi i g_m(s)}|-\rangle)/\sqrt{2} = X^{g_m(0^m)}(|+\rangle + e^{\frac{2\pi i}{2^m}} \sum_t c_t \bigoplus_{k=1}^{m} t_k s_k |-\rangle)/\sqrt{2}$, which does not depend on a_t. The point is that this situation is essentially the same as the one where $|\varphi_m\rangle$ is prepared by Q_n as described in Section 3.1, i.e., where we can only have the values $a_j(m) \oplus x_j$ for every $1 \le j \le n$ and we have to use them to prepare $|\varphi_m\rangle = (|+\rangle + e^{\frac{2\pi i}{2^m}|x|}|-\rangle)/\sqrt{2}$, which does not depend on $a_j(m)$. Thus, roughly speaking, we can construct R_m in a similar way to a part of Q_n.

A slight difference between these situations is that, in Q_n, it is very easy to prepare the values $a_j(m) \oplus x_j$ from the input bits x_j, but, in R_m, we need to consider a quantum circuit for computing the parity values $a_t \oplus \bigoplus_{k=1}^{m} t_k s_k$ from the input bits s_k, i.e., for the operation on $2^m + m - 1$ qubits defined as $|s\rangle \bigotimes_t |a_t\rangle \mapsto |s\rangle \bigotimes_t |a_t \oplus \bigoplus_{k=1}^{m} t_k s_k\rangle$ for any $s \in \{0,1\}^m$ and $a_t \in \{0,1\}$. If we have $m2^{m-1}$ *initialized* ancillary qubits, it is easy to construct an $O(m)$-depth quantum circuit for the operation using the following algorithm:

1. Prepare 2^{m-1} copies of s_k on the ancillary qubits for every $1 \le k \le m$ in parallel.
2. Compute the parity value $a_t \oplus \bigoplus_{k=1}^{m} t_k s_k$ for every $t \in \{0,1\}^m \setminus \{0^m\}$ in parallel.

To implement Step 1, we apply fan-out gates on $2^{m-1} + 1$ qubits, each of which can be decomposed into an $O(m)$-depth quantum circuit. Since it is easy to construct an $O(\log m)$-depth quantum circuit for PA_m using a binary tree structure, we can implement Step 2 using a parallel application of such circuits. If we replace the initialized ancillary qubits with uninitialized ones, the circuit does not work. However, applying the circuit again yields the desired values. In fact, the first circuit outputs $a_t \oplus \bigoplus_{k=1}^{m} t_k s_k \oplus d$ for some $d \in \{0,1\}$ that is computed from the (unknown) values in $\{0,1\}$ representing the initial state of the uninitialized ancillary qubits, and the second one outputs $a_t \oplus \bigoplus_{k=1}^{m} t_k s_k \oplus d \oplus d = a_t \oplus \bigoplus_{k=1}^{m} t_k s_k$ as desired. Using this circuit, we construct R_m and show the following lemma. The details can be found in the full version of the paper [21].

▶ **Lemma 8.** *The circuit R_m computes g_m. It uses no initialized ancillary qubit and $O(m2^m)$ uninitialized ancillary qubits, and its depth is $O(m^2)$, when the elementary gate set is $\mathcal{G}$.*

Combining R_m with Q_n immediately implies Theorem 1:

Proof of Theorem 1. By Lemmas 6, 7, and 8, we can use Q_n and R_m to implement the algorithm for $f_n \in \mathcal{S}_n$ described at the beginning of Section 3 and the whole circuit has the desired complexity. ◀

4 Classical Algorithms with Access to Shallow Quantum Circuits

Let $p(n)$ be a polynomial and C_n be a constant-depth quantum circuit on n qubits consisting of the gates in $\mathcal{G}$. The problem $\mathrm{MAT}(p(n), C_n)$ is to compute a real number α_x such that $|\alpha_x - |\langle 0^n | C_n | x\rangle|^2| \le 1/p(n)$ for any input $x \in \{0,1\}^n$. For any $x, w \in \{0,1\}^n$, we define $F_n(x,w) = \langle x | C_n^\dagger (\bigotimes_{j=1}^{n} Z_j^{w_j}) C_n | x\rangle$, where $w = w_1 \cdots w_n$ and Z_j is Z applied to the j-th qubit of C_n. As shown in [17], $\mathrm{MAT}(p(n), C_n)$ can be solved with probability exponentially (in n) close to 1 if there exists a probabilistic algorithm A_{F_n} such that, for any $x, w \in \{0,1\}^n$, the probability that $|A_{F_n}(x,w) - F_n(x,w)| \le 0.5/p(n)$ is exponentially close to 1. In fact, due to the Chernoff-Hoeffding bound, the algorithm for $\mathrm{MAT}(p(n), C_n)$ on input $x \in \{0,1\}^n$ is described with some $K = n^{O(1)}$ as follows: Choose $w(j) \in \{0,1\}^n$ uniformly at random and compute $A_{F_n}(x, w(j))$ for every $1 \le j \le K$, and output $(1/K) \sum_{j=1}^{K} A_{F_n}(x, w(j))$.

The probabilistic algorithm A_{F_n} in [17] can be considered as a repetition of a commuting quantum circuit D_{2n} for the Hadamard test with $2n$ input qubits and one output qubit. For any $x, w \in \{0,1\}^n$, the output of D_{2n} with the input qubits initialized to $|x\rangle|w\rangle$ and output qubit initialized to $|0\rangle$ is 0 with probability $(1 + F_n(x,w))/2$. Thus, when the outputs 0 and 1 are regarded as 1 and -1, respectively, due to the Chernoff-Hoeffding bound, A_{F_n} is described with some $L = n^{O(1)}$ as follows, where the input is the pair of x and w: Perform D_{2n} with the input qubits initialized to $|x\rangle|w\rangle$ and output qubit initialized to $|0\rangle$, and obtain its output $z_j(x,w) \in \{1,-1\}$ for every $1 \le j \le L$. After that, output $(1/L)\sum_{j=1}^{L} z_j(x,w)$.

We construct a parallelized version of the Hadamard test, denoted by E_{2n}, by using uninitialized ancillary qubits. Although the standard Hadamard test is a sequential application of controlled gates with the same control qubit, roughly speaking, E_{2n} first prepares the copies of the state of the control qubit on uninitialized ancillary qubits and then applies the gates in parallel by using the copies. To be precise, let $x = x_1 \cdots x_n, w = w_1 \cdots w_n \in \{0,1\}^n$. We prepare $2n$ input qubits $X_1, \ldots, X_n, W_1, \ldots, W_n$, one output qubit Y, and n uninitialized ancillary qubits $G(1), \ldots, G(n)$, where X_j, W_j, and Y are initialized to $|x_j\rangle$, $|w_j\rangle$, and $|0\rangle$, respectively. The initial state of the uninitialized ancillary qubits is arbitrary. The circuit E_{2n} is defined as follows:

1. Apply a H gate to Y.
2. Apply a fan-out gate on $n+1$ qubits to $G(1), \ldots, G(n)$, and Y, where Y is the control qubit.
3. Apply C_n to $X_1, \ldots, X_{n-1}$, and X_n.
4. Apply a 2-controlled Z gate to $G(j)$, X_j, and W_j for every $1 \le j \le n$ in parallel.
5. Apply $C_n^\dagger$ to $X_1, \ldots, X_{n-1}$, and X_n.
6. Apply the gates in Step 2 and Step 1 (in this order).

Each fan-out gate can be decomposed into an $O(\log n)$-depth quantum circuit. Moreover, a 2-controlled Z gate can be decomposed into a constant number of the gates in $\mathcal{G}$ [1, 18]. Thus, E_{2n} is an $O(\log n)$-depth circuit consisting of the gates in $\mathcal{G}$. It has the desired output probability distribution. The proof can be found in the full version of the paper [21].

▶ **Lemma 9.** *For any $x, w \in \{0,1\}^n$, the output of E_{2n} with the input qubits initialized to $|x\rangle|w\rangle$ and output qubit initialized to $|0\rangle$ is 0 with probability $(1 + F_n(x,w))/2$.*

This lemma immediately implies Theorem 2:

Proof of Theorem 2. We replace D_{2n} in the above-mentioned algorithm for $\mathrm{MAT}(p(n), C_n)$ with E_{2n}. By Lemma 9, the output probability distribution of E_{2n} is the same as that of D_{2n}. Thus, as with the original algorithm, the resulting algorithm solves $\mathrm{MAT}(p(n), C_n)$. ◀

5 Limitations of Uninitialized Ancillary Qubits

5.1 Our Idea for Proving Theorem 3

For any integer $s \ge 1$, an s-controlled Toffoli gate is decomposed into an s-controlled Z gate sandwiched between two H gates [8]. Thus, to prove Theorem 3, it suffices to consider an unbounded Z gate in place of an unbounded Toffoli gate. We assume on the contrary that there exists a depth-d quantum circuit C_n for PA_n with n input qubits, one output qubit, $p = O(\log n)$ initialized ancillary qubits, and $q = n^{O(1)}$ uninitialized ancillary qubits such that it consists of the gates in $\mathcal{G}$, unbounded fan-out gates on $(\log n)^{O(1)}$ qubits, and unbounded Z gates, where $d = O((\log n)^\delta)$ for some constant $0 \le \delta < 1$. When all unbounded Z gates in C_n act on a small number of qubits, such as $O(\log n)$ qubits, since d is sufficiently small, the proof of Bera [3] implies that there exists an input qubit of C_n such that the output of

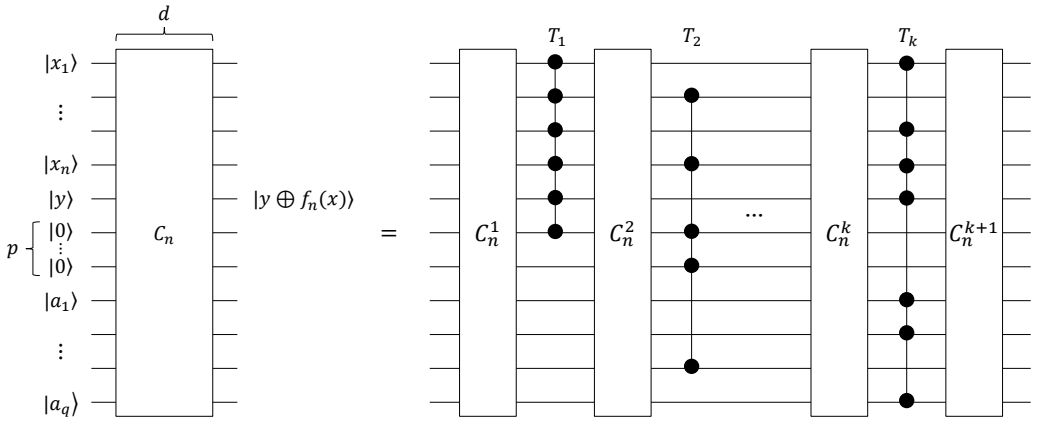


Figure 2 Circuit C_n for f_n and its decomposition. The initial states of the input qubits, output qubit, and uninitialized ancillary qubits are $|x_1\rangle \cdots |x_n\rangle$, $|y\rangle$, and $|a_1\rangle \cdots |a_q\rangle$, respectively, for any $x = x_1 \cdots x_n \in \{0,1\}^n$, $y \in \{0,1\}$, and $a_1 \cdots a_q \in \{0,1\}^q$. Gates $T_1, \ldots, T_k$ are unbounded Z gates.

C_n does not depend on the input qubit. Thus, C_n cannot compute PA_n since the output of PA_n changes if any one of the n input bits changes. This contradicts the assumption.

The remaining case is when there exists an unbounded Z gate on a large number of qubits. Let $\tilde{C}_n$ be the circuit obtained from C_n by removing all such gates. Bera [3] showed that, when C_n does not have any ancillary qubit, it is well approximated by $\tilde{C}_n$ in the sense that, when the state of the input qubits is a computational basis state chosen uniformly at random, the output of C_n coincides with that of $\tilde{C}_n$ with high probability. Since C_n computes PA_n, $\tilde{C}_n$ computes PA_n with high probability. Thus, we obtain a contradiction as in the above case since all gates in $\tilde{C}_n$ act on a small number of qubits. To apply this idea to our setting, we show that C_n with p initialized ancillary qubits and q uninitialized ancillary qubits in state $|a\rangle$ for some $a \in \{0,1\}^q$ is well approximated (in the sense described above) by $\tilde{C}_n$ with the same state. The former circuit computes PA_n since C_n with an arbitrary initial state of the uninitialized ancillary qubits computes PA_n. Thus, the latter circuit computes PA_n with high probability, and we obtain a contradiction as in the above simple case.

5.2 Analysis of a General Circuit and Its Application

We analyze a general depth-d quantum circuit C_n with n input qubits, one output qubit, and p initialized and q uninitialized ancillary qubits such that it consists of the gates in $\mathcal{G}$, unbounded fan-out gates, and unbounded Z gates. Its key property is described as follows:

▶ **Lemma 10** ([3, 2]). *Let C_n be a depth-d quantum circuit with n input qubits and one output qubit (possibly with ancillary qubits). If all gates in C_n act on at most w qubits, then the output of C_n can depend only on the states of at most w^d input qubits.*

Let $t \geq 2$ be an integer and $\mathcal{G}_t$ be the set of all unbounded Z gates in C_n that act on more than or equal to t qubits. We consider the case where $\mathcal{G}_t \neq \emptyset$ and assume that $\mathcal{G}_t = \{T_1, \ldots, T_k\}$ for some $k \geq 1$, where, for any $1 \leq l \leq k$, if T_l is in layer L of C_n, then T_{l+1} is in layer $L' \geq L$. We decompose C_n into the gates in $\mathcal{G}_t$ and the other parts as depicted in Fig. 2, where C_n computes a Boolean function f_n on n bits and C_n^j is a quantum circuit consisting of gates that are not in $\mathcal{G}_t$ for any $1 \leq j \leq k+1$. Such a decomposition is not unique in general, but the point is to fix a decomposition. For any $1 \leq l \leq k$, we define a quantum circuit V_l as follows: $V_1 = C_n^1$ and $V_l = C_n^l T_{l-1} V_{l-1}$ for any $2 \leq l \leq k$. We also

define $\Delta_l(x,y,b) = ||T_l V_l|x\circ y\circ b\rangle - V_l|x\circ y\circ b\rangle||$ and $\Delta(x,y,b) = ||C_n|x\circ y\circ b\rangle - \tilde{C}_n|x\circ y\circ b\rangle||$ for any $x \in \{0,1\}^n$, $y \in \{0,1\}$, and $b \in \{0,1\}^{p+q}$. Here, the symbol "$\circ$" represents the concatenation of bit strings, $|||v\rangle|| = \sqrt{\langle v|v\rangle}$ for any vector $|v\rangle$, and $\tilde{C}_n = C_n^{k+1}C_n^k\cdots C_n^2C_n^1$. Let U_n be a random variable uniformly distributed over $\{0,1\}^n$. We evaluate the probability $\Pr[\Delta(U_n,y,b) < \varepsilon]$ as follows. The proof can be found in the full version of the paper [21].

▶ **Lemma 11.** $\Pr[\Delta(U_n,y,b) < \varepsilon] \geq 1 - (k^2/\varepsilon^2)\sum_{l=1}^k \mathrm{E}[\Delta_l(U_n,y,b)^2]$ *for any* $\varepsilon > 0$, $y \in \{0,1\}$, *and* $b \in \{0,1\}^{p+q}$.

To evaluate the value $\sum_{l=1}^k \mathrm{E}[\Delta_l(U_n,y,b)^2]$, let t_l be the number of qubits on which T_l acts, $u_l = n+p+q+1-t_l$, and $t_{\min} = \min\{t_l | 1 \leq l \leq k\}$. We define $V_l|x\circ y\circ b\rangle = \sum_{i\in\{0,1\}^{t_l}}\sum_{j\in\{0,1\}^{u_l}} g^{(l)}_{x\circ y\circ b}(i\circ j)|i\circ j\rangle$ for any $x \in \{0,1\}^n$, $y \in \{0,1\}$, and $b \in \{0,1\}^{p+q}$, where $g^{(l)}_{x\circ y\circ b}(i\circ j)$ is a complex number. The qubits represented by $i \in \{0,1\}^{t_l}$ correspond to the qubits on which T_l acts. Of course, for any $1 \leq l \leq k$, T_l does not always act on the first t_l qubits in C_n. We therefore need to apply some permutation of all qubits; however, since such a permutation does not affect Lemma 13, which is the key to Theorem 3, we omit it.

We evaluate the above value as follows. The proof can be found in the full version [21].

▶ **Lemma 12.** $\sum_{l=1}^k \mathrm{E}[\Delta_l(U_n,y,b)^2] \leq k2^{p+q+3}/2^{t_{\min}}$ *for any* $y \in \{0,1\}$ *and* $b \in \{0,1\}^{p+q}$. *Moreover, there exists some* $a \in \{0,1\}^q$ *such that* $\sum_{l=1}^k \mathrm{E}[\Delta_l(U_n,0,0^p\circ a)^2] \leq k2^{p+3}/2^{t_{\min}}$.

Lemmas 11 and 12 immediately imply the following evaluation:

▶ **Lemma 13.** *There exists some* $a \in \{0,1\}^q$ *such that* $\Pr[\Delta(U_n,0,0^p\circ a) < \varepsilon] \geq 1 - k^3 2^{p+3}/(\varepsilon^2 2^{t_{\min}})$ *for any* $\varepsilon > 0$.

Lemmas 10 and 13 imply Theorem 3 as follows:

Proof of Theorem 3. We assume on the contrary that there exists a quantum circuit C_n for PA_n described in Section 5.1. Since $p = O(\log n)$, there exists a constant $c > 0$ such that $p \leq c\log n$ when n is sufficiently large. We define $t = (c+4)\log(n+p+q+1)$ and consider $\mathcal{G}_t$ described above. When $\mathcal{G}_t = \emptyset$, all gates in C_n act on at most $w = (\log n)^{O(1)}$ qubits. By Lemma 10, the output of C_n can depend only on the states of at most $w^d = o(n)$ input qubits. Thus, there exists an input qubit of C_n such that the output of C_n does not depend on the input qubit. This yields a contradiction as described in Section 5.1.

We consider the remaining case where $\mathcal{G}_t \neq \emptyset$. In this case, we apply the above analysis of a general circuit. It holds that $p \leq c\log n$, $k \leq (n+p+q+1)d/t_{\min}$, and $t_{\min} \geq (c+4)\log(n+p+q+1)$. Thus, by Lemma 13 with $\varepsilon = 0.1$,

$$\Pr[\Delta(U_n,0,0^p\circ a) < 0.1] \geq 1 - \left(\frac{d}{(c+4)\log(n+p+q+1)}\right)^3 \frac{800n^c}{(n+p+q+1)^{c+1}}$$

for some $a \in \{0,1\}^q$. Let us express this value on the right-hand side by $1-\gamma$. Thus, there exists a set $S \subseteq \{0,1\}^n$ such that S has at least $2^n(1-\gamma)$ elements and, for any $x \in S$, $\Delta(x,0,0^p\circ a) < 0.1$. Since γ goes to 0 as n goes to infinity, $2^n(1-\gamma) > 2^{n-1}$ when n is sufficiently large. A simple calculation shows that, for any $x \in \{0,1\}^n$ satisfying $\Delta(x,0,0^p\circ a) < 0.1$, the output of $\tilde{C}_n|x\circ 0\circ 0^p\circ a\rangle$ coincides with that of $C_n|x\circ 0\circ 0^p\circ a\rangle$ with probability of at least $1 - 0.1^2 = 0.99$ [3, 2]. When the initial state of the uninitialized ancillary qubits is $|a\rangle$, C_n computes PA_n. Thus, for any $x \in S$, the output of $\tilde{C}_n|x\circ 0\circ 0^p\circ a\rangle$ is $\mathrm{PA}_n(x)$ with probability of at least 0.99. This contradicts the fact obtained by the following argument. Since all gates in $\tilde{C}_n$ act on at most $(\log n)^{O(1)}$ qubits, as described for the case where $\mathcal{G}_t = \emptyset$, by Lemma 10, there exists an input qubit of $\tilde{C}_n$ such that the output of $\tilde{C}_n$ does not depend on the input qubit. This implies that, for at most 2^{n-1} elements $x \in \{0,1\}^n$, the output of $\tilde{C}_n|x\circ 0\circ 0^p\circ a\rangle$ is $\mathrm{PA}_n(x)$ with probability greater than 0.5. ◀

References

1. A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457–3467, 1995.
2. D. Bera. *Quantum circuits: power and limitations*. PhD thesis, Boston University, 2010.
3. D. Bera. A lower bound method for quantum circuits. *Information Processing Letters*, 111(15):723–726, 2011.
4. S. Bravyi, D. Gosset, and R. König. Quantum advantage with shallow circuits, 2017. arXiv:1704.00690.
5. H. Buhrman, R. Cleve, M. Koucký, B. Loff, and F. Speelman. Computing with a full memory: catalytic space. In *Proceedings of the 46th ACM Symposium on Theory of Computing (STOC)*, pages 857–866, 2014.
6. R. Cleve and J. Watrous. Fast parallel circuits for the quantum Fourier transform. In *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 526–536, 2000.
7. D. P. DiVincenzo. The physical implementation of quantum computation. *Fortschritte der Physik*, 48(9–11):771–783, 2000.
8. M. Fang, S. Fenner, F. Green, S. Homer, and Y. Zhang. Quantum lower bounds for fanout. *Quantum Information and Computation*, 6(1):46–57, 2006.
9. S. Fenner, F. Green, S. Homer, and Y. Zhang. Bounds on the power of constant-depth quantum circuits. In *Proceedings of Fundamentals of Computation Theory (FCT)*, volume 3623 of *Lecture Notes in Computer Science*, pages 44–55, 2005.
10. F. Green, S. Homer, C. Moore, and C. Pollett. Counting, fanout, and the complexity of quantum ACC. *Quantum Information and Computation*, 2(1):35–65, 2002.
11. P. Høyer and R. Špalek. Quantum fan-out is powerful. *Theory of Computing*, 1(5):81–103, 2005.
12. S. Jukna. *Boolean Function Complexity: Advances and Frontiers*. Springer, 2012.
13. E. Knill and R. Laflamme. Power of one bit of quantum information. *Physical Review Letters*, 81(25):5672–5675, 1998.
14. T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. O'Brien. Quantum computing. *Nature*, 464:45–53, 2010.
15. G. De las Cuevas, W. Dür, M. van den Nest, and M. A. Martin-Delgado. Quantum algorithms for classical lattice models. *New Journal of Physics*, 13(093021), 2011.
16. C. Moore and M. Nilsson. Parallel quantum computation and quantum codes. *SIAM Journal on Computing*, 31(3):799–815, 2001.
17. X. Ni and M. van den Nest. Commuting quantum circuits: efficient classical simulations versus hardness results. *Quantum Information and Computation*, 13(1&2):54–72, 2013.
18. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.
19. Y. Takahashi and N. Kunihiro. A quantum circuit for Shor's factoring algorithm using $2n+2$ qubits. *Quantum Information and Computation*, 6(2):184–192, 2006.
20. Y. Takahashi and S. Tani. Collapse of the hierarchy of constant-depth exact quantum circuits. *Computational Complexity*, 25(4):849–881, 2016.
21. Y. Takahashi and S. Tani. Power of uninitialized qubits in shallow quantum circuits, 2017. arXiv:1608.07020v3.
22. Y. Takahashi, T. Yamazaki, and K. Tanaka. Hardness of classically simulating quantum circuits with unbounded Toffoli and fan-out gates. *Quantum Information and Computation*, 14(13&14):1149–1164, 2014.
23. J. Watrous. On the complexity of simulating space-bounded quantum computations. *Computational Complexity*, 12(1–2):48–84, 2003.

Lower Bounds on Black-Box Reductions of Hitting to Density Estimation

Roei Tell
Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel
roei.tell@weizmann.ac.il

Abstract

Consider a deterministic algorithm that tries to find a string in an unknown set $S \subseteq \{0,1\}^n$, under the promise that S has large density. The only information that the algorithm can obtain about S is *estimates of the density of* S in adaptively chosen subsets of $\{0,1\}^n$, up to an additive error of $\mu > 0$. This problem is appealing as a derandomization problem, when S is the set of satisfying inputs for a circuit $C : \{0,1\}^n \to \{0,1\}$ that accepts many inputs: In this context, an algorithm as above constitutes a deterministic black-box reduction of the problem of *hitting* C (i.e., finding a satisfying input for C) to the problem of *approximately counting* the number of satisfying inputs for C on subsets of $\{0,1\}^n$.

We prove tight lower bounds for this problem, demonstrating that naive approaches to solve the problem cannot be improved upon, in general. First, we show a tight trade-off between the estimation error μ and the required number of queries to solve the problem: When $\mu = O(\log(n)/n)$ a polynomial number of queries suffices, and when $\mu \geq 4 \cdot (\log(n)/n)$ the required number of queries is $2^{\Theta(\mu \cdot n)}$. Secondly, we show that the problem "resists" parallelization: Any algorithm that works in iterations, and can obtain $p = p(n)$ density estimates "in parallel" in each iteration, still requires $\Omega\left(\frac{n}{\log(p)+\log(1/\mu)}\right)$ iterations to solve the problem.

This work extends the well-known work of Karp, Upfal, and Wigderson (1988), who studied the setting in which S is only guaranteed to be non-empty (rather than dense), and the algorithm can only probe subsets for the *existence* of a solution in them. In addition, our lower bound on parallel algorithms affirms a weak version of a conjecture of Motwani, Naor, and Naor (1994); we also make progress on a stronger version of their conjecture.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness, Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Approximate Counting, Lower Bounds, Derandomization, Parallel Algorithms, Query Complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.58

Related Version A full version of this paper is available on ECCC [8].

Funding This research was partially supported by the Minerva Foundation with funds from the Federal German Ministry for Education and Research.

Acknowledgements The author thanks his advisor, Oded Goldreich, for clearly defining the problem studied in this paper, and for many useful discussions and ideas. The author also thanks Karthik C. S. for useful discussions and for many valuable comments that improved the presentation of the paper. The author also thanks anonymous reviewers for helpful comments that improved the presentation of the paper.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 58; pp. 58:1–58:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

If we want to catch a lion in the desert, a binary search is the method of choice: We bipartition the desert, check in which cell the lion resides, and recurse. But what if we want to catch a lion in a lions den, where lions are in abundance?

We are interested in the following problem. A deterministic algorithm tries to find a string in an unknown set $S \subseteq \{0,1\}^n$, where it is a-priori guaranteed that the density of S is large (e.g., $|S| \geq 2^{n-1}$). The only information that the algorithm can obtain about S is an *estimation* of the density of S in any subset $Q \subseteq \{0,1\}^n$ of its choice. That is, for any subset $Q \subseteq \{0,1\}^n$, the algorithm can obtain a value $\tilde{\nu}(Q)$ such that $\tilde{\nu}(Q) = \frac{|Q \cap S|}{|Q|} \pm \mu$, for a small $\mu > 0$. Can the algorithm find a string $s \in S$ more efficiently than simply going over all singletons in $\{0,1\}^n$ and checking whether or not each of them is in S?

As noted by Goldreich [1, Thm. 3.5], if the estimation error μ is sufficiently small, then the problem can be solved efficiently, using a method similar to the method of conditional probabilities. Specifically, the algorithm iteratively constructs a string $s \in S$ bit-by-bit, where in each iteration the algorithm decides which value for the next bit would yield a higher density of S in the resulting subcube, up to an error of μ. Unfortunately, this method requires that the error μ will be inversely proportional to the number of iterations (i.e., $\mu = 1/O(n)$). Moreover, the method has the drawback of being *inherently sequential*: Constructing an n-bit solution involves sequentially solving n decision problems. Thus, our main question in this work is the following:

> Can the problem be solved more efficiently, compared to the naive "equipartition and recurse" algorithm? Specifically, can we improve the dependency on the estimation error, and can a parallel algorithm solve the problem faster?

The problem that we study in this work is especially relevant in the context of *derandomization*. Think of S as the set of satisfying inputs for some circuit $C : \{0,1\}^n \to \{0,1\}$. Two fundamental problems in derandomization are the problem of *hitting* the set S (i.e., finding a satisfying input for C), and the problem of *approximately counting* the size of S (i.e., estimating the acceptance probability of C; this problem is sometimes called the *Circuit Acceptance Probability Problem*). From this perspective, the question underlying the current work is the following: Does the hitting problem for a circuit C reduce to the approximate counting problem for C (on subsets of $\{0,1\}^n$), in general? [1] And more specifically, can we improve on the sequential reduction that was presented above if we are given only limited "non-black-box" information about C?

The problem that we study can also be viewed as an extension of two classical problems. Specifically, consider the problem of reducing the search for a string in a non-empty set S to the task of deciding, for any $Q \subseteq \{0,1\}^n$, whether or not $Q \cap S \neq \emptyset$ (see [6]). Our problem is a natural extension of this problem to the setting where the target set S is *dense*, with the corresponding decision task adapted accordingly (to estimating the *density* of S in any subset Q). Moreover, our problem is a variant of an open problem of Motwani, Naor and Naor [7]: They also considered a dense set S, but in their setting the algorithm can obtain the *exact* density of S in any subset $Q \subseteq \{0,1\}^n$, rather than a density estimation. Indeed,

[1] In typical settings, if one can approximate the acceptance probability of C, then one can also approximate the acceptance probability of C on subcubes of $\{0,1\}^n$. Our main lower bounds hold even if the algorithm can approximate the acceptance probability of C on *any* subset of $\{0,1\}^n$, whereas the upper bounds only rely on such estimations on subcubes.

their setting is also natural, but less relevant to derandomization than our setting. They conjectured that in their setting, even if the algorithm can obtain, in each step, the density of S in $\text{poly}(n)$ sets in parallel, still no significant speed-up over the method of conditional probabilities is possible (for details see Section 2). As far as we know, no progress has been made on their conjecture prior to this work.

1.1 Lower bounds on parallel algorithms

The first question that we consider is whether the problem described above can be solved faster by a parallel algorithm. Specifically, assume that the algorithm searching for a string in S works in *iterations*. In each iteration, the algorithm sends $p = p(n)$ queries to a *density oracle*, where each question is a set $Q_i \subseteq \{0,1\}^n$, and then receives p answers, where each answer is a density estimation $\tilde{\nu}(Q_i) = \frac{|Q_i \cap S|}{|Q_i|} \pm \mu$. Can such an algorithm find a string $s \in S$ using less than n iterations?

If μ is small, one can obtain an efficient algorithm that uses $n/\log(p)$ iterations, by a straightforward adaptation of the method of conditional probabilities: Instead of constructing a solution bit-by-bit, construct a solution block-by-block, where each block consists of $\log(p)$ bits. This yields the following upper bound.

▶ **Theorem 1.** *(an upper bound for parallel algorithms; informal). For any $p \in \mathbb{N}$ and $\mu < \frac{\log(p+1)}{4n}$, an algorithm that uses p density estimations with error μ in each iteration can find a string in an unknown set S of size $|S| \geq 2^{n-1}$ using $\frac{n}{\log(p+1)}$ iterations.* [2]

Our main result for the setting of parallel algorithms is that, unless the estimation error is extremely small (i.e., unless $\mu = o(1/\text{poly}(p))$), the algorithm described above essentially cannot be improved upon. In particular, for the natural setting of $p = \text{poly}(n)$ and $\mu = 1/\text{poly}(n)$, the problem requires an almost-linear number of iterations to solve.

▶ **Theorem 2.** *(a lower bound on the number of iterations of parallel algorithms; informal). For any $\mu > 0$ and $p \in \mathbb{N}$, algorithms that use p density estimations with error μ in each iteration need at least $\frac{n}{\log(p+1)+\log(1/\mu)}$ iterations to find a string in a set S of size $|S| \geq 2^{n-1}$.*

The lower bound in Theorem 2 is proved under the assumption that $|S| \geq 2^{n-1}$. One might expect that if S has significantly larger density (e.g., $|S| = (1 - o(1)) \cdot 2^n$), then an algorithm might be able to find a string in S using less iterations. Our second result shows that even if $|S| \geq (1 - 2^{-\Omega(n)}) \cdot 2^n$, then the lower bound asserted in Theorem 2 still essentially holds. Actually, we generalize Theorem 2, by showing a trade-off between the density of S and a lower bound on the number of iterations required to find a string in S.

▶ **Theorem 3.** *(a lower bound for parallel algorithms and large sets; informal). For any $0 < \mu \leq 1/2$, and $p \in \mathbb{N}$, and $\epsilon > 0$, algorithms as above need at least $\frac{\epsilon \cdot n}{\log(p+1)+\log(1/\mu)}$ iterations in order to find a string in an unknown set S of size $|S| \geq 2^n - 2^{\epsilon \cdot n}$.*

Recall that we are particularly interested in this problem when S is the set of satisfying inputs for some circuit $C : \{0,1\}^n \to \{0,1\}$, and the algorithm tries to find a satisfying input for C by estimating the acceptance probability of C in subsets of $\{0,1\}^n$. In this setting one does not necessarily expect the algorithm to be able to estimate the acceptance probability

[2] To obtain an upper bound of $n/\log(p+1)$, instead of $n/\log(p)$, the algorithm partitions the space in each iteration into $p+1$ sets, and relies on the estimations for the density of S in the first p sets in order to estimate the density of S in the $(p+1)^{th}$ set.

of C in very "complex" subsets. When we impose such a limitation on the circuit complexity of the queries that the algorithm makes, we obtain the following strengthening of Theorem 3, which asserts that the same lower bound holds even if the algorithm is guaranteed that S can be decided by a relatively simple circuit.

▶ **Theorem 4.** *(the lower bound for parallel algorithms holds even for "simple" circuits; informal). Assume that for any query $Q \subseteq \{0,1\}^n$ that the algorithm from Theorem 3 makes, the set Q can be decided by a circuit from a circuit class $\mathcal{C}$. Then, the lower bound in Theorems 3 holds even if the algorithm is guaranteed that the set S can be decided by a conjunction of negations of $n \cdot p$ circuits from $\mathcal{C}$.*

One corollary of Theorem 4 is that if the algorithm only estimates the acceptance probability of C on *subcubes* of $\{0,1\}^n$, then even the guarantee that C is a polynomial-sized CNF does not allow to bypass the lower bound in Theorem 3.

Note that the size of the circuit for S in Theorem 4 is larger than the total number of queries made by the algorithm. This is no coincidence: If the algorithm is given the size of C, and is allowed to make a number of queries that is polynomial in the size of C, then the algorithm can simply query, in parallel, the singletons in the output-set of a pseudorandom generator for C (assuming that such a generator exists; see, e.g., [12, Prop 7.8]). In contrast, Theorem 4 asserts that when the number of queries is *smaller* than the size of the circuit, and the algorithm uses density estimations (rather than only query singletons), the guarantee that C is a conjunction of negations of $n \cdot p$ circuits from $\mathcal{C}$ does not suffice in order to bypass the lower bound in Theorem 3.

When S is the set of satisfying inputs for a circuit C, the setting of $|S| = (1 - o(1)) \cdot 2^n$ corresponds to circuits that accept almost all of their inputs. This setting, called *quantified derandomization*, has recently been introduced by Goldreich and Wigderson (see [2, 9, 10]).

1.2 Lower bounds on algorithms with large estimation error

The second question that we consider in this paper is what happens when the estimation error μ is too large to use the method of conditional probabilities (i.e., $\mu = \omega(1/n)$). In this setting, we are not necessarily interested in parallel algorithms, but simply ask what is the *number of estimations* needed in order to find a string $s \in S$.

Similarly to the previous section, we show that a naive algorithmic approach essentially cannot be improved upon. Specifically, consider an algorithm that works in iterations; in each iteration, the algorithm equipartitions the "current" search space into $2^{4\mu \cdot n}$ sets, obtains estimates for the density of S in each of the sets, and recurses into the set in which S has the highest estimated density. Since the depth of the recursion tree is less than $1/4\mu$, an estimation error of μ suffices for this algorithm. [3] This yields the following upper bound.

▶ **Theorem 5.** *(an upper bound for algorithms with large estimation errors; informal). For any error $\mu > 0$, an algorithm that only uses density estimations with error μ can find a string in an unknown set S of size $|S| \geq 2^{n-1}$ using less than $\frac{2}{\mu} \cdot 2^{4\mu \cdot n}$ density estimations.*

Note that when the estimation error is $\mu = O(\log(n)/n)$, the algorithm described above uses $\mathrm{poly}(n)$ density estimations. Our main result in the current section is that whenever $\mu \geq \frac{4 \cdot \log(n)}{n}$, the upper bound in Theorem 5 is essentially tight. To see this, observe that

[3] Indeed, when $\mu = O(\log(n)/n)$, this is essentially the same algorithm as the parallel algorithm from Theorem 1; the number of estimations in each iteration is $p = 2^{4 \cdot \mu \cdot n} = \mathrm{poly}(n)$.

when $\mu \geq \frac{4 \cdot \log(n)}{n}$, the upper bound in Theorem 5 is $\frac{2}{\mu} \cdot 2^{4\mu \cdot n} = 2^{O(\mu \cdot n)}$; and when $\mu > 1/4$, the upper bound exceeds 2^n. Thus, it is nearly matched by the following lower bound.

▶ **Theorem 6.** *(a lower bound on the number of estimations needed by algorithms with large estimation errors; informal). For any error $\mu \geq \frac{4 \cdot \log(n)}{n}$, at least $2^{\Omega(\mu \cdot n)}$ density estimations are needed in order to find a string in an unknown set S of size $|S| \geq 2^{n-1}$. Moreover, if $\mu \geq \frac{1}{4} + \Omega(1)$, then $\Omega(2^n/n)$ estimations are needed to find a string in S of size $|S| \geq 2^{n-1}$.*

Theorem 6 implies in particular that if the error satisfies $\mu = \omega(\log(n)/n)$, then the problem cannot be solved efficiently (i.e., using only $\text{poly}(n)$ estimations). We also generalize Theorems 5 and 6, by showing a trade-off between the density of S, denoted by ρ, and the number of estimations required to find a string in S. This generalization is most interesting when considering sets S with small density (e.g., density $\rho = O(\mu)$), in which case the problem is much more difficult.

▶ **Theorem 7.** *(a general trade-off between density, error, and the number of queries needed to solve the problem; informal). For any error $\mu \geq 12 \cdot \log(n)/n$, the following holds:*

1. *For any density $\rho \leq (2 - \Omega(1)) \cdot \mu$, at least $2^n/\text{poly}(n)$ density estimations are needed to find a string in a set S of size $|S| \geq \rho \cdot 2^n$.*
2. *For any density $\rho \geq (2 - o(1)) \cdot \mu$, it holds that $2^{\Theta((\mu/\rho) \cdot n)}$ density estimations are necessary and sufficient to find a string in a set S of size $|S| \geq \rho \cdot 2^n$.*

1.3 Lower bounds on algorithms with *no* estimation error

Finally, we consider the setting suggested by Motwani, Naor, and Naor [7], in which there is no estimation error (i.e., $\mu = 0$). They conjectured that a naive "equipartition and recurse" will still be essentially optimal in this setting.

We focus on the question of the required number of queries to solve the problem (and leave open the question of parallelism). Recall that the algorithm obtains density values, and not binary answers, and thus a naive information-theoretic lower bound of n queries does not hold. Nevertheless, we show that $n - O(1)$ queries are still necessary.

▶ **Theorem 8.** *(the minimal number of* exact *density queries; informal). Consider algorithms that, for an unknown set S of size $|S| > \rho \cdot 2^{n-1}$, can query an oracle to obtain the exact density of S in any subset $Q \subseteq \{0,1\}^n$. Then, the number of queries that such algorithms need in order to find a string $s \in S$ is at least $n - \lfloor \log(1/(1-\rho)) \rfloor - 1$.*

The lower bound in Theorem 8 is tight, up to a single bit. The proofs of Theorem 8 and of the corresponding upper bound appear in [8, Sec. 7].

1.4 Organization

In Section 2 we discuss the context of our results and previous related work. In Section 3 we explain the techniques used to obtain our results, in high-level. Section 4 contains the formal definitions of the algorithms described above. In Section 5 we prove the lower bounds on solving the problem in parallel (i.e., Theorems 2 and 3), and the corresponding upper bound (i.e., Theorem 1) is proved in [8, Apdx. A]. In Section 6 we prove the lower bounds on solving the problem when the estimation error is large (i.e., Theorems 6 and 7), and the corresponding upper bounds (i.e., Theorem 5 and the upper bound in Item (2) of Theorem 7) are proved in [8, Apdx. A].

2 Background and previous works

Several years ago, Goldreich [1] considered the hypothesis that *promise-$\mathcal{BPP}$ = promise-$\mathcal{P}$*, which in particular implies that the density of satisfying inputs for a given circuit can be estimated in polynomial time. Goldreich showed that it follows that *$\mathcal{BPP}$ search problems* (see the definition in [1, Sec. 3]) can also be solved in deterministic polynomial time, and in particular, there exists a deterministic polynomial-time algorithm that finds a satisfying input for a given circuit that accepts most of its inputs. Indeed, his reduction of search problems to problems of estimating the density of solutions in subsets of the search space is based on the method of conditional probabilities. The current work can be viewed as an extension of his study, which considers a more generic reduction of the task of finding a satisfying input for a circuit C to estimating the density of satisfying inputs for C in subsets of the domain, [4] and asks whether his solution for this problem can be improved upon, in general, with only limited "non-black-box" information about the circuit C.

Thus, this work is situated within a line of works that study the limitations of "black-box" techniques in derandomization. Although many of the best current derandomization results are based on constructions that are essentially black-box (i.e., on pseudorandom generators that use very little information about the circuit that they wish to "fool"), black-box techniques nevertheless have disadvantages in the context of derandomization. In particular, constructing a black-box pseudorandom generator (or a hitting-set generator) for a circuit class necessitates proving strong corresponding lower bounds against that circuit class; [5] and black-box techniques cannot be used in certain settings for hardness amplification, which is a common strategy to try and prove the lower bounds necessary to construct pseudorandom generators via the hardness-randomness paradigm (see, e.g., [13, 11]).

The current work also extends the study of Karp, Upfal, and Wigderson [6], who proved lower bounds for the setting in which the target set S is only guaranteed to be non-empty (rather than dense), and the algorithm can obtain, for any set $Q \subseteq \{0,1\}^n$, an answer to whether or not a solution exists in Q (i.e., whether or not $Q \cap S = \emptyset$). The authors showed that the "equipartition-and-recurse" strategy is optimal in this setting, since any strategy requires $n/\log(p+1)$ iterations to find a solution, in general, where p is the number of decision problems that can be solved in parallel in each iteration.

Motwani, Naor, and Naor [7] considered a setting in which S can be dense (rather than just non-empty), and conjectured that a similar lower bound would hold in this setting even if the algorithm trying to find a string $s \in S$ can obtain, for any $Q \subseteq \{0,1\}^n$, the exact number of solutions in Q (i.e., the value $|S \cap Q|$). Thus, Theorems 2 and 3 affirm a weak version of their conjecture, where the difference is that in our case, instead of obtaining the exact number of solutions in Q, the algorithm can only obtain an estimate of the number of solutions in Q. In addition, Theorem 8 proves their conjecture for the special case of $p = 1$ (i.e., when there is no parallelism).

3 Our techniques

To understand the challenge, let us first recall the techniques of Karp, Upfal, and Wigderson [6]. They proved their lower bound (for algorithms that can only probe for the existence of a

[4] That is, we only consider the hypothesis that one can efficiently estimate the density of satisfying inputs for C on subsets of its domain, rather than the hypothesis that *promise-$\mathcal{BPP}$ = promise-$\mathcal{P}$*.

[5] Analogous implications of *any* derandomization of a circuit class (i.e., not necessarily a black-box one) are known, but the implied lower bounds are weaker and less direct (see, e.g., [3, 4, 5, 14]).

solution in any subset) by an adversarial argument: For any algorithm A, they simulated the execution of A, and supplied adversarial answers, in order to delay A's progress in finding $s \in S$. In their argument, in each iteration, the adversary has to answer A's queries in a manner that is consistent with the current information available to A (i.e., with all previous answers). However, since the adversary only provides "yes/no" answers, relatively little information is revealed about S in each iteration, and thus relatively few constraints are imposed upon the adversary when engineering answers in subsequent iterations.

As noted by [7], it is not a-priori clear how to extend the foregoing strategy to a setting in which A obtains the *exact* number of solutions in any subset that it queries. This is the case because in the latter setting, the adversary has to answer A's queries with exact density values that are *perfectly consistent* with some fixed set S; this requirement imposes strict constraints on the adversary in each iteration. This seems to require much more careful engineering of answers on the adversary's part.

The key observation underlying our lower bounds is that if we, as adversaries, are allowed a small error in our answers to A, then we do not need to engineer the answers so carefully. One approach to take advantage of this relaxed setting, which we use in the proofs of Theorems 2 and 3, is to start the simulation with a "tentative" set S, modify this set adversarially throughout the execution, and answer A in each iteration according to the current state of S (instead of engineering artificial answers). If the modifications that we make to the set S throughout the execution are not too substantial, then the final version of S is not very different from any of the tentative ones, which implies that the error in our answers was never too big. This approach also allows us to show that there exists a relatively simple circuit that decides S (i.e., to obtain Theorem 4).

An alternative approach, which we use in the proofs of Theorems 6 and 7, is to answer A's queries according to some set of rules, and in the end construct *an adversarial distribution* of sets such that a set sampled from the distribution will be consistent with our answers, up to a small error, with high probability. That is, the fact that we are allowed a small error allows us to avoid the explicit construction of a single adversarial set, and instead rely on an adversarial distribution of sets. Specifically, we will answer each query $Q \subseteq \{0,1\}^n$ of A only according to the size of Q; and in the end we will construct a distribution $\mathcal{S}_A$ over sets $S \subseteq \{0,1\}^n$, which depends on the specific queries that A issued, such that a set $S \sim \mathcal{S}_A$ will be consistent with our answers, up to an error of μ, with high probability.

In contrast, in Theorem 8 we consider algorithms without an estimation error, and thus we indeed need to fully engineer exact adversarial answers. To do so, we maintain a template for the set S, which is a partition of $\{0,1\}^n$ such that each set P in the partition is labeled with the density of S in P. Given each query of A, we refine the partition, while making sure that unless the partition is extremely refined, no set P in the partition is fully contained in S. Thus, the algorithm needs to use many queries, in order to yield an extremely refined partition, which will allow it to find a singleton $s \in S$.

4 Preliminaries

All logarithms in the paper are to base 2. We formally define the algorithms described in Section 1 by using the notion of oracle machines, where the oracle is the device supplying density estimations. An oracle function for a set S gets as input a sequence of p density queries, and outputs p estimations for the density of S in each of the queried sets, where each estimation is correct up to a relative additive error of μ.

▶ **Definition 9.** (p-parallel μ-error density estimators). For $n, p \in \mathbb{N}$, and $\mu < 1$, and a

set $S \subseteq \{0,1\}^n$, a function $f_S : \mathcal{P}(\{0,1\}^n)^p \to [0,1]^p$ is called a p-parallel μ-error density estimator for S if for every $\vec{Q} = (Q_1, Q_2, ..., Q_p) \in \mathcal{P}(\{0,1\}^n)^p$, and every $j \in [p]$, it holds that $\left|\tilde{\nu}(Q_j) - \frac{|Q_j \cap S|}{|Q_j|}\right| \leq \mu$, where $\tilde{\nu}(Q_j)$ is the j^{th} element in the sequence $f_S(\vec{Q})$. [6]

▶ **Definition 10.** (hitters with access to density estimators). Let $\mu : \mathbb{N} \to [0,1)$, and let $p : \mathbb{N} \to \mathbb{N}$. A deterministic algorithm A is called a hitter with oracle access to p-parallel μ-error density estimators if for every $n \in \mathbb{N}$ and $S \subseteq \{0,1\}^n$, given input 1^n and oracle access to a $p(n)$-parallel $\mu(n)$-error density estimator for S, the algorithm A outputs a string $s \in S$.

We deliberately avoid the question of how A specifies its queries to the oracle, and just assume that all queries can be perfectly communicated. Our lower bounds are thus solely information-theoretic. On the other hand, the algorithms establishing the upper bounds in the paper only use queries about subcubes of $\{0,1\}^n$, which can be easily communicated in any reasonable model of an oracle Turing machine.

When $p = 1$ (i.e., when there is no parallelism), we just refer to a `$\mu$-error density estimator` and to a `hitter with oracle access to $\mu$-error density estimators`. A hitter as in Definition 10 operates in iterations, where in each iteration it issues a query-tuple to the oracle (i.e., a sequence of p sets), and receives an answer-tuple (i.e., p correponsing density estimations). When we will discuss a specific query (resp., specific answer), we will usually mean one of the p sets queried in some iteration (resp., one of the p density estimations given in the answer).

5 Lower bounds on parallel algorithms

Let us begin by stating Theorem 2 and proving it. For simplicity, we will prove a lower bound of $\frac{n}{\log(p)+\log(1/\mu)+1}$, instead of $\frac{n}{\log(p+1)+\log(1/\mu)}$. [7]

▶ **Theorem 11.** *(a lower bound for parallel algorithms; Theorem 2, restated). For $\mu : \mathbb{N} \to (0, \frac{1}{2})$ and $p : \mathbb{N} \to \mathbb{N}$, let A be a* hitter with oracle access to p-parallel μ-error density estimators. *Then, for any $n \in \mathbb{N}$, there exists a set $S \subseteq \{0,1\}^n$ of size $|S| \geq 2^{n-1}$ and a $p(n)$-parallel $\mu(n)$-error density estimator f_S for S such that the number of iterations that A uses when given oracle access to f_S is at least*

$$\frac{n}{\log(p(n)) + \log(1/\mu(n)) + 1} .$$

Proof overview. Let $n \in \mathbb{N}$, and let $p = p(n)$ and $\mu = \mu(n)$. Assume towards a contradiction that A always uses $R < \frac{n}{\log(p)+\log(1/\mu)+1}$ iterations. We will construct a set S and a p-parallel μ-error density estimator f_S that "fool" A: That is, f_S answers all of A's queries in a manner that is consistent with S, up to a relative error of μ, but in the end of the execution of A, the algorithm outputs a string that is not in S.

It will be more convenient to work with a definition for hitters that is slightly different from the one in Definition 10: Instead of requiring that the algorithm outputs a string $s \in S$ in the end of the execution, we require that the hitter will ask the oracle about a singleton $Q = \{s\}$, and receive an answer $\tilde{\nu}(\{s\}) > \mu$ (which implies that $s \in S$). The number of iterations required to solve the problem is identical in both definitions, up to ± 1 iteration.

We will simulate the execution of A, and answer the algorithm's queries adversarially, in order to delay its progress in finding small sets that contain elements in S. Specifically,

[6] We denote by $\mathcal{P}(\{0,1\}^n)$ the power set of $\{0,1\}^n$.
[7] The proof of the slightly tighter lower bound appears in [8, Thm. 11].

let us call a set Q a *positive set* if at some point during the execution of A, the algorithm queried the oracle about the set Q and was answered by a non-zero value (i.e., by $\tilde{\nu}(Q) > 0$). Our goal is that in the end of the execution, all positive sets will be of size at least 2, which will imply that A did not find any positive singleton.

Our answers throughout the execution of A have to be consistent with some fixed set S, up to an estimation error of μ. To ensure this, we will maintain a "tentative" version of the set S, and provide answers in each iteration according to the tentative set at that iteration. We will show that the final version of the set S, which is the tentative set in the end of the execution, is not very different from any of the non-final versions. Thus, the answers given to A are consistent, up to a small error (less than μ), with the set S.

We initialize the tentative set as $S_0 = \{0,1\}^n$. In iteration $i \in [R]$, we modify the current tentative set, denoted S_{i-1}, and obtain the new tentative set, S_i, as follows:

- We start iteration i with a guarantee that all positive sets are of size at least h_i (where h_i is a parameter to be determined).
- Given A's queries, we remove from the tentative set all the strings from queried-sets that are "too small". That is, the new tentative set S_i is obtained by removing from S_{i-1} every string that belongs to a queried set Q' such that $|Q'| < \ell_i$ (where ℓ_i is also a parameter to be determined).
- We answer the queries of A according to the (current) tentative set S_i; that is, the query Q is answered by $|Q \cap S_i|/|Q|$. Thus, in the next iteration, all positive sets will be of size at least $h_{i+1} = \ell_i$.

We define S to equal the tentative set at the end of the execution (i.e., $S = S_R$). Let us now describe our setting of parameters, and explain why it suffices to prove the theorem. In the first iteration we have $h_1 = 2^n$, which holds vacuously. We define ℓ_i in each iteration such that $\ell_i/h_i = \mu/(2 \cdot p)$. It follows that $\ell_R = (\mu/(2p))^R \cdot h_1 = (2p/\mu)^{-R} \cdot 2^n$. Relying on the hypothesis that $R < \frac{n}{\log(p)+\log(1/\mu)+1} = \log_{2p/\mu}(2^n)$, we have that $\ell_R > 1$, which means that A did not find any positive singleton.

Now, let Q be a positive set that was queried in iteration i, and let us count the number of strings removed from the tentative set in subsequent iterations. The number of strings removed in iteration $i+1$ is less than $p \cdot \ell_{i+1} = (\mu/2) \cdot h_{i+1} = (\mu/2) \cdot \ell_i$. Since in iteration i we answer positively only for sets of cardinality at least ℓ_i, we know that $|Q| \geq \ell_i$, and thus the number of strings removed in iteration $i+1$ is less than $(\mu/2) \cdot |Q|$. In subsequent iterations, the number of strings removed from the tentative set decays exponentially (see [8, Claim 11.4]); hence, when summing over iterations $i+1, ..., R$, the overall number of strings removed is less than $\mu \cdot |Q|$. Similarly, the overall number of strings removed from $S_0 = \{0,1\}^n$ is less than $\mu \cdot |S_0|$, and thus we have that $|S| > 2^{n-1}$. For full details see [8, Thm. 11]. ◀

We now prove Theorem 3, which asserts a trade-off between the density of S and a lower bound on the number of iterations needed to find a string $s \in S$.

▶ **Theorem 12.** *(a lower bound for parallel algorithms and large sets; Theorem 3, restated). For $\mu : \mathbb{N} \to (0, \frac{1}{2})$ and $p : \mathbb{N} \to \mathbb{N}$, let A be a hitter with oracle access to p-parallel μ-error density estimators. Then, for any $\epsilon > 0$ and $n \in \mathbb{N}$, there exists* a set $S \subseteq \{0,1\}^n$ of size $|S| \geq 2^n - 2^{\epsilon \cdot n}$ *and a $p(n)$-parallel $\mu(n)$-error density estimator f_S for S such that the number of iterations that A uses when given oracle access to f_S is at least $\frac{\epsilon \cdot n}{\log(p(n))+\log(1/\mu(n))+1}$.* [8]

[8] For proof of the slightly stronger lower bound $\frac{\epsilon \cdot n}{\log(p+1)+\log(1/\mu)}$ see [8, Thm. 12].

Proof overview. In the proof of Theorem 11, the threshold that defines a "small" query starts out quite high; that is, $\ell_1 = \alpha \cdot 2^n$, where $\alpha = (\mu/2p)$. (In each subsequent iteration, the threshold decreases by a multiplicative factor of α.) Thus, in the first iteration we might remove $\ell_1 \cdot p \approx \mu \cdot 2^n$ strings from the tentative set.

In the current proof we wish to avoid the removal of so many strings from the tentative set. To do so, we will set the threshold *in the first iteration* to be about $2^{\epsilon \cdot n}$. Specifically, denoting by $\mathcal{R} = \frac{n}{\log(p)+\log(1/\mu)+1}$ the number of iterations in the proof of Theorem 2, we will set the threshold ℓ_1 to the value that it obtained (in the proof of Theorem 2) in iteration $i \approx (1-\epsilon) \cdot \mathcal{R}$; that is, $\ell_1 \approx \alpha^{(1-\epsilon)\cdot\mathcal{R}} \cdot 2^n = 2^{\epsilon \cdot n}$. Then, in each subsequent iteration, we will decrease the threshold by a multiplicative factor of α. The number of removed strings will thus be less than $2^{\epsilon \cdot n}$, whereas the number of iterations (i.e., the lower bound) will be $\epsilon \cdot \mathcal{R} = \frac{\epsilon \cdot n}{\log(p)+\log(1/\mu)+1}$. For full proof details, see [8, Thm. 12]. ◀

The circuit complexity of the "hard" set S.

Recall that a motivating example for the problem discussed in this paper is when S is the set of satisfying inputs for some circuit C, and the algorithm A tries to find a satisfying input for C. One might intuitively expect that in order to construct a "hard" set S for A (i.e., a set S that forces A to use many iterations), a "complicated" circuit C will be needed. However, we observe that the circuit complexity of the "hard" sets that are constructed in the proofs of Theorems 11 and 12 is proportional only to the circuit complexity of the sets that A queried. Thus, we can now prove Theorem 4, which asserts that the lower bound in Theorem 12 holds even if the algorithm is guaranteed that S can be decided by a relatively simple circuit:

▶ **Theorem 13.** *(the lower bound on parallel algorithm holds even for "simple" sets; Theorem 4, restated). Let μ, p, and A be as in Theorem 12, and further assume that for every query $Q \subseteq \{0,1\}^n$ that A makes, the set Q can be decided by a circuit from a circuit class $\mathcal{C}$. Then, for any $\epsilon > 0$ and $n \in \mathbb{N}$, there exists* a set $S \subseteq \{0,1\}^n$ of size $|S| \geq 2^n - 2^{\epsilon \cdot n}$ that can be decided by an conjunction of negations of at most $n \cdot p$ circuits from $\mathcal{C}$ *and a $p(n)$-parallel $\mu(n)$-error density estimator f_S for S such that the number of iterations that A uses when given oracle access to f_S is at least* $\frac{\epsilon \cdot n}{\log(p(n))+\log(1/\mu(n))+1}$. [9]

Proof. The set S that is constructed in the proof of Theorem 12 is obtained by starting from the tentative set $S_0 = \{0,1\}^n$, and removing subsets that correspond to some of the algorithm's queries (i.e., the ones that were deemed "too small" in the relevant iteration). Thus, the set S is the intersection of the complements of at most $(\epsilon \cdot \mathcal{R}) \cdot p < n \cdot p$ subsets such that each subset can be decided by a circuit from $\mathcal{C}$. ◀

6 Lower bounds on algorithms with large estimation error

We now formally state Theorem 6, and prove the first part of the theorem; for the "moreover" part, see [8, Prop. 14]. In fact, we will prove a statement slightly stronger than the one in the first part of Theorem 6: Instead of proving the lower bound when the set S of size $|S| \geq 2^{n-1}$, we will prove it assuming that $|S| \geq (1-\mu) \cdot 2^n$.

▶ **Theorem 14.** *(a lower bound for large errors; Theorem 6, restated). Let $\mu : \mathbb{N} \to (0, 1/2)$ such that $\mu(n) \geq \frac{4 \cdot \log(n)}{n}$, and let A be a hitter with oracle access to μ-error density estimators.*

[9] For proof of the slightly stronger lower bound $\frac{\epsilon \cdot n}{\log(p+1)+\log(1/\mu)}$ see [8, Thm. 13].

Then, for any sufficiently large $n \in \mathbb{N}$, there exists a set $S \subseteq \{0,1\}^n$ of size $|S| \geq (1-\mu) \cdot 2^n$, and a μ-error density estimator f_S for S, such that the number of iterations that A uses when given oracle access to f_S is at least $2^{\Omega(\mu \cdot n)}$.

Proof overview. Let $\mu' = \mu/2$, and for simplicity, assume that μ is constant. Similar to the proofs in Section 5, we simulate A and provide adversarial answers. However, in the current proof our answers will be given according to one fixed rule, which does not change throughout the execution (in contrast to the previous proofs, in which the "threshold" for defining small sets decreased in each iteration).

The main idea is to arrange all sets of size more than n in "levels", where the i^{th} level contains sets of size between $2^{(i-1)\cdot\mu'\cdot n}$ and $2^{i\cdot\mu'\cdot n}$. Then, during the execution, we will output the same fixed estimate for all queried-sets in the same level; specifically, for sets in level i, we will output the estimate $i \cdot \mu'$. Note that if we use $1/\mu'$ levels, then our estimate for the set $\{0,1\}^n$, which is in the highest level, is $\tilde{\nu}(\{0,1\}^n) = 1$; this implies that any S consistent with our answers has density at least $1-\mu$. To see that there exists a set S that is consistent with such answers, fix a queried-set Q in level i. If the algorithm makes at most $R \ll 2^{\mu'\cdot n}$ queries, then the vast majority of strings in Q do not belong to queried-sets of level $i-2$ or less, since the number of such strings is at most $R \cdot 2^{(i-2)\cdot\mu'\cdot n} \ll |Q|$. Thus, if we include every string $w \in Q$ in S with probability $\ell(w) \cdot \mu'$, where $\ell(w)$ is the level of the smallest queried-set that contains w, then the vast majority of strings in Q will be included in S with probability either $(i-1)\cdot\mu'$ or $i \cdot \mu'$. Hence, the expected density of S in Q will be close to the interval $\left[(i-1)\cdot\mu', i\cdot\mu'\right]$, which implies that, with high probability, the actual density of S in Q will not deviate from our estimate of $\tilde{\nu}(Q) = i \cdot \mu'$ by more than $2 \cdot \mu' = \mu$.

Let us now provide further details for this idea. We partition the power set of $\{0,1\}^n$ into levels as follows. The zero level, denoted $\mathcal{L}_0$, consists of all sets of size at most (roughly) n. For $i = 1, ..., 1/\mu'$, the i^{th} level, denoted $\mathcal{L}_i$, consists of all sets that are not included in any level $j < i$, and that are of size at most $2^{i\cdot\mu'\cdot n}$. Observe that for every $i \geq 3$, every set in $\mathcal{L}_i$ is larger than every set in $\mathcal{L}_{i-2}$ by a multiplicative factor of at least $2^{\mu'\cdot n}$. After $R \approx \mu' \cdot 2^{\mu'\cdot n}$ iterations, in which we act as above (i.e., answer $\tilde{\nu}(Q) = i \cdot \mu'$ for $Q \in \mathcal{L}_i$), we let S be a random set such that every string $w \in \{0,1\}^n$ is included in S, independently, with probability $\ell(w) \cdot \mu'$ (recall that $\ell(w)$ is the level of the smallest queried-set that contains w, or $\ell(w) = 1/\mu'$, if w was not included in any queried-set).

Note that A cannot find a positive singleton, since we output the estimate zero for every queried-set in $\mathcal{L}_0$ (i.e., every set of size smaller than (roughly) n). Also note that strings in queried-sets in $\mathcal{L}_0$ are never included in S, and thus our estimate (of zero) for every queried-set in $\mathcal{L}_0$ is always correct. Our main claim is that, with high probability, *for any queried-set $Q \in \mathcal{L}_i$, where $i \geq 1$, the density of S in Q, denoted by $\delta_S(Q)$, satisfies* $i \cdot \mu' - \mu \leq \delta_S(Q) \leq i \cdot \mu' + \mu$. This claim implies that our estimate for Q is correct, up to an error of μ. Let us sketch the proof of this claim.

- To see that $\delta_S(Q) \leq i \cdot \mu' + \mu = (i+2) \cdot \mu'$, note that every $w \in Q$ is included in S with probability $\ell(w) \cdot \mu' \leq i \cdot \mu'$, where the inequality is because $\ell(w)$ is upper bounded by the level of Q, which is i.
- To see that $\delta_S(Q) \geq i \cdot \mu' - \mu = (i-2) \cdot \mu'$, first note that this lower bound is trivial for $i \leq 2$ (because $i - 2 \leq 0$). If $i \geq 3$, the number of strings from queried-sets of level $j \leq i-2$ in Q is at most $R \cdot \max_{Q' \in \mathcal{L}_{i-2}} |Q'|$. Now, recall that every set in $\mathcal{L}_i$ is larger than every set in $\mathcal{L}_{i-2}$ by a multiplicative factor of at least $2^{\mu'\cdot n}$, and that $R \approx \mu' \cdot 2^{\mu'\cdot n}$. Hence, the vast majority of strings $w \in Q$ satisfy $\ell(w) \geq i-1$, and they will be included in S with probability at least $(i-1)\cdot\mu'$.

The foregoing establishes that a random set is consistent with our answers to A. Since we

output the estimate 1 for any set in level $i = 1/\mu'$, and in particular for the set $\{0,1\}^n \in \mathcal{L}_{1/\mu'}$, it follows that the overall density of S is at least $(1-\mu)\cdot 2^n$. For full proof details, which also handle a sub-constant error μ, see [8, Thm. 15]. ◀

We now prove the lower bound in Item (2) of Theorem 7, which generalizes Theorem 6, by asserting a trade-off between the density of S and the number of estimations required to find a string in it. The proof of Item (1) of Theorem 7 appears in [8, Prop. 14].

▶ **Theorem 15.** *(a lower bound for large errors and arbitrary density; Theorem 7, restated). Let $\mu : \mathbb{N} \to (0, 1/2)$ such that $\mu(n) \geq \frac{12\cdot\log(n)}{n}$, and let A be a hitter with oracle access to μ-error density estimators. Then, for any sufficiently large $n \in \mathbb{N}$, and any density $\rho \in [\mu(n), 1-\mu(n)]$, there exists a set $S \subseteq \{0,1\}^n$ of size $|S| \geq \rho \cdot 2^n$, and a μ-error density estimator f_S for S, such that the number of iterations that A uses when given oracle access to f_S is at least $2^{\Omega((\mu/\rho)\cdot n)}$.*

Proof overview. The proof is obtained by modifying the proof of Theorem 6. In the proof of Theorem 6, we partitioned the collection of sets of size larger than (roughly) n into $1/\mu'$ levels. The density of the set S was proportional to the *number of levels*, because we supplied the density estimate $i \cdot \mu'$ for sets in level i (and in particular, the estimate 1 for the set $\{0,1\}^n$ in level $i = 1/\mu'$). On the other hand, the lower bound on the number of estimates was proportional to the *height of each level* $i = 2, 3, ..., 1/\mu'$, where the *height* of a level is ratio between the size of the largest set in it and the size of the smallest set in it.

In the current proof, we wish to obtain a set S with smaller density (i.e., density ρ instead of density $1-\mu$), but improve the lower bound on the number of estimates. We will do so by partitioning the subsets of $\{0,1\}^n$ into fewer levels, each of larger height. Specifically, we let $\mu' = \mu/2$, and partition the collection of sets of size more than n into slightly more than ρ/μ' levels, each of height about $2^{n/(\rho/\mu')}$. Using an analysis very similar to that in the proof of Theorem 6, we will obtain a lower bound of about $2^{(\mu'/\rho)\cdot n}$ estimates, and the set S will be of density about $(\rho/\mu')\cdot\mu' = \rho$. For full proof details, see [8, Thm. 16]. ◀

References

1. Oded Goldreich. In a world of p=bpp. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, pages 191–232. Springer, 2011. doi:10.1007/978-3-642-22670-0_20.
2. Oded Goldreich and Avi Wigderson. On derandomizing algorithms that err extremely rarely. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 109–118, 2014. doi:10.1145/2591796.2591808.
3. R. Impagliazzo and A. Wigderson. Randomness vs. time: De-randomization under a uniform assumption. In *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 734–, 1998.
4. Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences*, 65(4):672–694, 2002.
5. Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.
6. Richard M. Karp, Eli Upfal, and Avi Wigderson. The complexity of parallel search. *Journal of Computer and System Sciences*, 36(2):225–253, 1988.

7 Rajeev Motwani, Joseph Naor, and Moni Naor. The probabilistic method yields deterministic parallel algorithms. *Journal of Computer and System Sciences*, 49(3):478–516, 1994.

8 Roei Tell. Lower bounds on black-box reductions of hitting to density estimation. *Electronic Colloquium on Computational Complexity: ECCC*, 23:50, 2016.

9 Roei Tell. Improved bounds for quantified derandomization of constant-depth circuits and polynomials. In *Proc. 32nd Annual IEEE Conference on Computational Complexity (CCC)*, pages 18:1–18:49, 2017.

10 Roei Tell. Quantified derandomization of linear threshold circuits. *Electronic Colloquium on Computational Complexity: ECCC*, 24:145, 2017.

11 Luca Trevisan and Salil P. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity*, 16(4):331–364, 2007.

12 Salil P. Vadhan. *Pseudorandomness*. Foundations and Trends in Theoretical Computer Science. Now Publishers, 2012.

13 Emanuele Viola. The complexity of constructing pseudorandom generators from hard functions. *Computational Complexity*, 13(3-4):147–188, 2005.

14 Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM Journal of Computing*, 42(3):1218–1244, 2013.